UNIVERSITI TEKNOLOGI MALAYSIA

PSZ 19:16 (Pind. 1/07)

DECLARATION OF THESIS / UNDERGRADUATE PROJECT PAPER AND COPYRIGHT

Author's full name : MOHAMAD AMIN BIN NOORDIN

Date of birth : 19 FEBRUARY 1989

Title : REMOTE VIDEO SURVEILLANCE USING FPGA

Academic Session: 2012/2013

I declare that this thesis is classified as :

☐	**CONFIDENTIAL**	(Contains confidential information under the Official Secret Act 1972)*
☐	**RESTRICTED**	(Contains restricted information as specified by the organization where research was done)*
☑	**OPEN ACCESS**	I agree that my thesis to be published as online open access (full text)

I acknowledged that Universiti Teknologi Malaysia reserves the right as follows:

1. The thesis is the property of Universiti Teknologi Malaysia.
2. The Library of Universiti Teknologi Malaysia has the right to make copies for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Certified by :

SIGNATURE	**SIGNATURE OF SUPERVISOR**
890214-07-5567	ASSOC.PROF.DR. MUHAMMAD NASIR BIN IBRAHIM
(NEW IC NO. /PASSPORT NO.)	**NAME OF SUPERVISOR**
Date :	Date :

NOTES : * If the thesis is CONFIDENTAL or RESTRICTED, please attach with the letter from the organization with period and reasons for confidentiality or restriction.

"I hereby declare that I have read this thesis and in my opinion
this thesis is sufficient in terms of scope and quality for the
award of the degree of Bachelor of Electronics Engineering"

Signature : _____

Name of Supervisor : ASSOC. PROF. DR. MUHAMMAD NASIR BIN IBRAHIM

Date : 24th June 2013

REMOTE VIDEO SURVEILLANCE USING FPGA

MOHAMAD AMIN BIN NOORDIN

A thesis awarded in fulfilment of the
requirement for the award of
Bachelor Degree in Electronics Engineering

Faculty of Electrical Engineering
Universiti Teknologi Malaysia

June 2013

"I hereby declare that this thesis entitled "Remote Video Surveillance Using FPGA" is the result of my own research except as cited in the references. The thesis has not been accepted for any degree and is not concurrently submitted in candidature of any other degree."

Signature : _____

Name : MOHAMAD AMIN BIN NOORDIN

Date : 24th June 2013

Dedicated to my beloved mother, brother, and sister and in the memories of my father.

ACKNOWLEDGEMENT

First and foremost I would like to express my greatest gratitude to my supervisor, Assoc. Prof. Dr Muhammad Nasir bin Ibrahim for the guidance, enthusiasm, and motivation given throughout the progress of this project. Without his continuous support and interest, this project would not be accomplished as presented here.

My sincere appreciation also goes to my mother, brother, and sister who has been supportive and has given me encouragement either morally or financially which enabled me to pursue on finishing the project.

I would also like to thank my friends who has directly or indirectly helped me in giving opinions, ideas, and other useful information throughout this project.

Nevertheless, my appreciation also goes to Mr. Jeevan Srikunan who willing to spend some of his time to help me with his technical skills and knowledge.

ABSTRACT

Technological innovations in imaging and video processing have revolutionized the video surveillance industry. The transition from analogue CCTV camera surveillance to network video surveillance which emphasizes the use of Ethernet is becoming more and more prominent. The network video surveillance system is also becoming more important. Currently, CCTV camera is used mostly in video surveillance with the network video surveillance slowly growing in importance. CCTV cameras need pc's to store data and it doesn't have its own storage, and unable to process high image quality. IP camera requires a lot bandwidth, so it limits its capability to stream. Microcontroller is cheaper than FPGA but it cannot beat FPGA in terms of processing speed and performance. It is also not flexible because any changes in the design will result in fabrication of new PCB since it is hard-wired. It is also 50 to 100 times slower in performance/watt compared to FPGA. To solve this problem a network video surveillance using FPGA DE2 has been developed. The main design was created in terms of block design before divided into hardware, software part, and image creation and storing part. The hardware part was created using SOPC builder and the software part was created using Nios II IDE software. The hardware part was only creating the interfaces between the components used. The software part involves the protocol creation, which is TCP, IP and MAC address setting, communication for DM9000A, and http implementation for requesting file from server to send to client requesting it. The image creation part is to create an html file with image embedded in it and store it in flash memory. The image file used in this project was created in an html formal and stored in flash memory using the DE2 control panel. The hardware creation, coding (software part), and image storing in flash memory was a success, but the code cannot be built because of software problem of not having the MicroC/OS-II component to run this web server type system.

ABSTRAK

Perkembangan teknologi di dalam pemprosesan imej dan video telah merevolusikan industri pemantauan video. Perubahan daripada pemantauan video analog camera CCTV kepada video jaringan yang menekankan penggunaan Ethernet menjadi semakin penting. Camera CCTV memerlukan komputer untuk menyimpan data dan tidak mempunyai penyimpanan data tersendiri, dan tidak boleh memproses imej yang berkualiti tinggi. Camera IP memerlukan jalurlebar yang besar, justeru mengurangkan kebolehan untuk pengaliran videonya. Mikropengawal adalah lebih murah daripada FPGA tetapi ia tidak boleh mengalahkan FPGA daripada segi kelajuan proses and prestasi. Ia juga tidak fleksibel kerana sebarang perubahan di dalam rekaan akan menyebabkan fabrikasi semula PCB kerana ia adalah rekaan berwayar. Ia juga 50 ke 100 kali ganda lebih perlahan daripada FPGA dari segi prestasi/watt. Untuk menyelesaikannya, satu jaringan pemantauan video telah direka dengan menggunakan FPGA DE2. Rekaan dibahagikan kepada dua iaitu bahagian perkakasan, perisian, dan penciptaan dan penyimpanan imej. Bahagian perkakasan direka dengan menggunakan SOPC Builder dan bahagian perisian direka dengan menggunakan Nios II IDE. Bahagian perkakasan hanya mereka bentuk pengantaramuka diantara komponen yang digunakan. Bahagian perisian computer melibatkan reka bentuk protocol iaitu TCP, penetapan alamat IP dan MAC, kominikasai untuk DM9000A, dan pelaksanaan http. Fail dicipta dalam bentuk hmtl dan imej disimpan dalamnya sebelum menyimpannya di dalam memori flash. Fail imej yang digunakan dalam projek ini direka dalam bentuk html dan disimpan di dalam memori flash dengan menggunakan panel kawalan DE2. Reka bentuk perkakasan elektronik, kod, dan penyimpanan imej ke dalam memori flash telah Berjaya dilaksanakan tetapi kod tersebut tidak dapat di bina kerana terdapat masalah dengan perisian komputer di mana ia tidak mempunyai komponen MicroC/OC-II untuk menjalankan system server web ini.

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE
	DECLARATION	
	ACKNOWLEDGEMENT	
	ABSTRACT	
	TABLE OF CONTENTS	
	LIST OF TABLES	
	LIST OF FIGURES	
	LIST OF ABBREVIATIONS	
CHAPTER 1	**INTRODUCTION**	
	1.1 Background of study	1
	1.2 Problem Statement	2
	1.3 Objective	3
	1.4 Scope	3
CHAPTER 2	**LITERATURE REVIEW**	
	2.1 FPGA	5
	2.2 Nios II Processor	7
	2.3 TCP/IP Protocol	9
	2.4 Ethernet	11
	2.5 Universal Serial Bus (USB)	13
	2.6 Related Works	17

2.6.1 The Design of Remote Image Monitoring System Based on DM9000A 17
2.6.2 A Reconfigurable Hardware Networking Platform for Smart Grid 18
2.6.3 Secure Remote Reconfiguration of an FPGA-based Embedded System 20

CHAPTER 3 RESEARCH METHODOLOGY

3.1 Work Flow 22
3.2 Project Design 23
3.2.1 Hardware Design 23
3.2.2 Software Design 24
3.2.3 Image File Creation and Storing 24
3.3 TCP vs UDP 24
3.4 Block Diagram of Design 26
3.5 Operation Flow 27

CHAPTER 4 RESULTS AND DISCUSSIONS

4.1 Image File 29
4.1.1 Image File Creation 29
4.1.2 Image File Storing 31
4.2 Hardware 33
4.3 Software 38
4.3.1 The Transport Layer Internet Protocol (IP) 38
4.3.2 The Nios II Coding 41
4.3.2.1 IP Address and MAC Address Setting 42
4.4 Expected Result 43
4.5 Problem and Troubleshooting 44

CHAPTER 5 CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions 47

5.2 Recommendations 48

REFERENCES 49

APPENDIX A 51

APPENDIX B 52

AAPENDIX C 53

LIST OF TABLES

TABLE NO.	TITLE	PAGE
2.1	C Code vs VHDL Implementation	19
4.1	TCP vs UDP	41

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE
2.1	The DE2 Board	6
2.2	Block Diagram of DE2 Board	6
2.3	Nios II Processor	8
2.4	Cat5 Cable with RJ45 Connector	11
2.5	Block Diagram for DM9000A	12
2.6	Fast Ethernet interface schematic inside FPGA	13
2.7	Type A	14
2.8	Type B	14
2.9	USB (ISP1362) Host and Device Schematic	15
2.10	ISP1362 Architectural Diagram	16
2.11	Hardware Principe Diagram	17
2.12	The connection of DM9000A with Nios II CPU	18
2.13	Proposed FPGA Based Data Communication for Smart Grid	19
2.14	Entity Communication Channels	21
3.1	Block Design of System	27
3.2	Operation Flow	28
4.1	index.html	30
4.2	not_found.html	30
4.3	DE2 Control Panel	31
4.4	Flash Memory Erasing	31
4.5	Write to Flash Memory	32
4.6	SOPC Hardware Design	33
4.7	Nios II Processor Type	34

4.8	Nios II Processor Instruction and Data Bus Setting	34
4.9	Nios II JTAG Setting	35
4.10	Flash Memory Data and Address Width	35
4.11	Flash Memory Timing	36
4.12	Red LED Setting	36
4.13	Green LED Setting	36
4.14	Button Setting	37
4.15	Switch Setting	37
4.16	DM9000A Custom Component Creation	38
4.17	Wireshark Protocol Analysis	39
4.18	Wireshark Protocol Hierarchy Statistics	40
4.19	DE2 Switches	42
4.20	LCD Display of MAC and IP Address	43
4.21	Image Display on Browser	44
4.22	MicroC/OS-II Missing Component	45
4.23	System Library Properties	45

LIST OF ABBREVIATIONS

FPGA	-	Field Programmable Gate Array
PCB	-	Printed Circuit Board
IP	-	Internet Protocol
TCP	-	Transfer Control Protocol
UDP	-	User Datagram Protocol
HDL	-	Hardware Description Language
LE	-	Logic Element
GPIO	-	General Input Output Port
RTOS	-	Real-Time Operating System
MMU	-	Memory Management Unit
EDS	-	Embedded Design Suite
ACK	-	Acknowledgement
SMTP	-	Simple Mail Transfer Protocol
FTP	-	File Transfer Protocol
HTTP	-	Hypertext Transfer Protocol
LAN	-	Local Area Network
CSMA/D	-	Carrier Sense Multiple Access with Collision Detection
MAC	-	Media Access Control
PHY	-	Physical Layer of OSI Model
USB	-	Universal Serial Bus
OTG	-	On The Go
SOPC	-	System On Programmable Chip
CMOS	-	Complementary Metal Oxide Semiconductor
ICMP	-	Internet Control Message Protocol
ARP	-	Address Resolution Protocol

ES	-	Embedded System
µIP	-	Micro IP
LWIP	-	Light Weight IP
SSDP	-	Simple Service Discovery Protocol
DHCP	-	Dynamic Host Control Protocol
SMB	-	Server Message Block
IGMP	-	Internet Group Management Protocol
DNS	-	Domain Name Service
BOOTP	-	Bootstrap Protocol

LIST OF APPENDICES

APPENDIX	TITLE	PAGE
APPENDIX A	FYP 1 Gantt Chart	51
APPENDIX B	FYP 2 Gantt Chart	52
APPENDIX C	Nios II Coding	53

CHAPTER 1

INTRODUCTION

1.1 Background of Study

The video surveillance trend is changing. Technological innovations in imaging and video processing have revolutionized the video surveillance industry. The transition from analogue CCTV camera surveillance to network video surveillance which emphasizes the use of Ethernet is becoming more and more prominent. The network video surveillance system is also becoming more important and having better quality. With this trend, the network video surveillance is going to replace analogue surveillance industry. The advances in this surveillance technology are tightly coupled with the advances in image sensing technologies and in signal processing capabilities. On the other hand, video surveillance using FPGA can be said as the next generation of video surveillance.

The field-programmable gate array (FPGA) is actually a semiconductor device that can be programmed after manufacturing. Instead of being restricted to any predetermined hardware feature, FPGA allows for reconfiguration of hardware even after the products has been installed in the field. By using FPGA, high performance of video processing can be achieved. Moreover, the data captured by the FPGA can also be sent through the Ethernet port and viewed by any computer on the same network. The low cost FPGAs are now making it possible to

implement high performance processing systems on a cost-effective and low-powered FPGA. Furthermore, this low-cost FPGA enables high performance signal processing. It has abundant multipliers, large amounts of on-chip memory, and with fast fabric performance. This is what makes the FPGA an ideal platform for the rapidly expanding field of video surveillance system.

1.2 Problem Statement

Currently, CCTV camera is used mostly in video surveillance with the network video surveillance slowly growing in importance. For the CCTV camera, it has a sensor which captures images. The resolution of the camera is limited to 720x575 pixels. For the CCTV camera, data cannot be directly accessed or taken from the camera. It needs a computer medium for storage of data. FPGA on the other hand can act as a stand-alone network video surveillance system. IP cameras which are now gaining in attention from people are a type of networked video surveillance. It requires more bandwidth than analog CCTV cameras, so this may limit its capability for streaming or recording. FPGA on the other hand has greater bandwidth capability, and will not have any limitations due to bandwidth. Moreover, many IP cameras are underpowered. For example, in dark conditions, the image processor spends more time trying to compensate for low light images and cannot output the same higher frame rate from normal lighting condition. FPGA enables high performance signal or video processing due to the availability of abundant multipliers and large amount of on-chip memory

Some people may question the use of FPGA instead of microcontroller in video surveillance. Microcontroller is cheaper than FPGA but it cannot beat FPGA in terms of processing speed and performance. FPGA has a lot of advantages over microcontroller. One of its biggest advantage is flexibility and reconfigurable. FPGA can always be reprogrammed by simply changing the Verilog code and reprogram the board. The hard-wired printed circuit board (PCB) is very hard to reconfigure once the PCB already done. Any changes in the design will result in

the fabrication of a new PCB. Furthermore microcontroller will not have enough processing power in video processing and telecommunications applications, especially the data path. Microcontroller is also not good in image processing where it will process image at a very slow time, and at very low quality. In those sorts of applications, a direct hard-wired logic is needed to process the packet. FPGA has larger memory on-chip than microcontroller, which means a greater bandwidth. FPGA also has a built in Ethernet module for network/ip communication. Moreover FPGA has a faster processing speed compare to any microcontroller and it also enables image compression and storage. FPGA can provide approximately 50 to 100 times the performance/watt of power consumed than that of a microprocessor. This makes it a better choice in terms of power compared to microprocessors.

1.3 Objective

This project has three main objectives. First is to create an image file and convert it in format so that it can be stored into the FPGA DE2 flash memory. The next objective is to create an Internet Protocol (IP) module for data and packets sending. The final objective is to view the image stored inside DE2 flash memory from a computer through the network.

1.4 Scope

A complete remote video surveillance system requires both the camera module and also the network module. For the camera module, the purpose is to store image captured using camera into the DE2 board flash memory. In order to use a USB

camera, the ISP1362 chip need to be utilized and protocols for this communication must be created. For the network module, the image stored inside the FPGA DE2 board must be able to be accessed when a computer request to view it. The scope reflects the aim of this research. Therefore, one of this project aims is to use TCP/UDP protocol to communicate between the FPGA DE2 and computer through a specific IP address set for the FPGA DE2 board. Other than that, the scope of this project is to use C language and Verilog to code for the interface of DM9000A chip and its protocol, programmed through Quartus II and Nios II EDS suite. The computer also must be able to access the FPGA DE2 flash memory, and view the image stored in from the computer.

CHAPTER 2

LITERATURE REVIEW

2.1 FPGA

Field Programmable Gate Array abbreviated as FPGA is an integrated circuit that is designed in a way that it can be configured by the user after manufacturing. FPGA has the capability to perform any logical function. The flexibility and reconfigurable ability of FPGA makes it a unique and preferred platform for project development. The configurability or the programming of the FPGA is usually done using hardware description language (HDL) such as VHDL or Verilog. An FPGA contains programmable logic components which are called logic elements (LEs) and a hierarchy of interconnects which are reconfigurable that allow the LEs to be connected physically. The FPGA can be reconfigured to perform complex combinational logic functions, flip-flops, and complete memory blocks or just simple AND and OR logics. There are several companies producing their own FPGAs. The top two companies are Altera and Xilinx. FPGA not only allow for digital programming, but it also has many features which allows user to implement a wide range of design. Take Altera FPGA DE2 as an example. The DE2 also has a lot of ports such as VGA, Ethernet, RS-232, USB 2.0, PS/2, two GPIO ports and also IrDA transceiver. This is what makes FPGA to be implemented on a wide range of design scope.

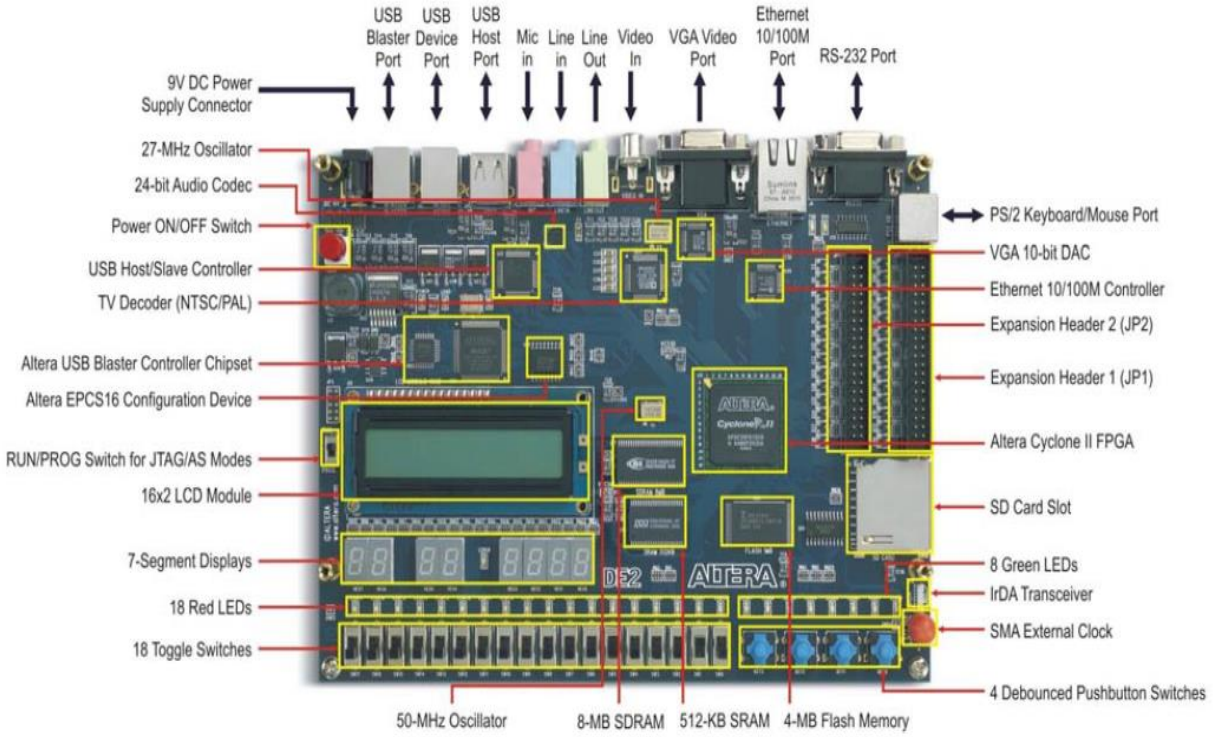


Figure 2.1: The DE2 board.(Corporation 2006)

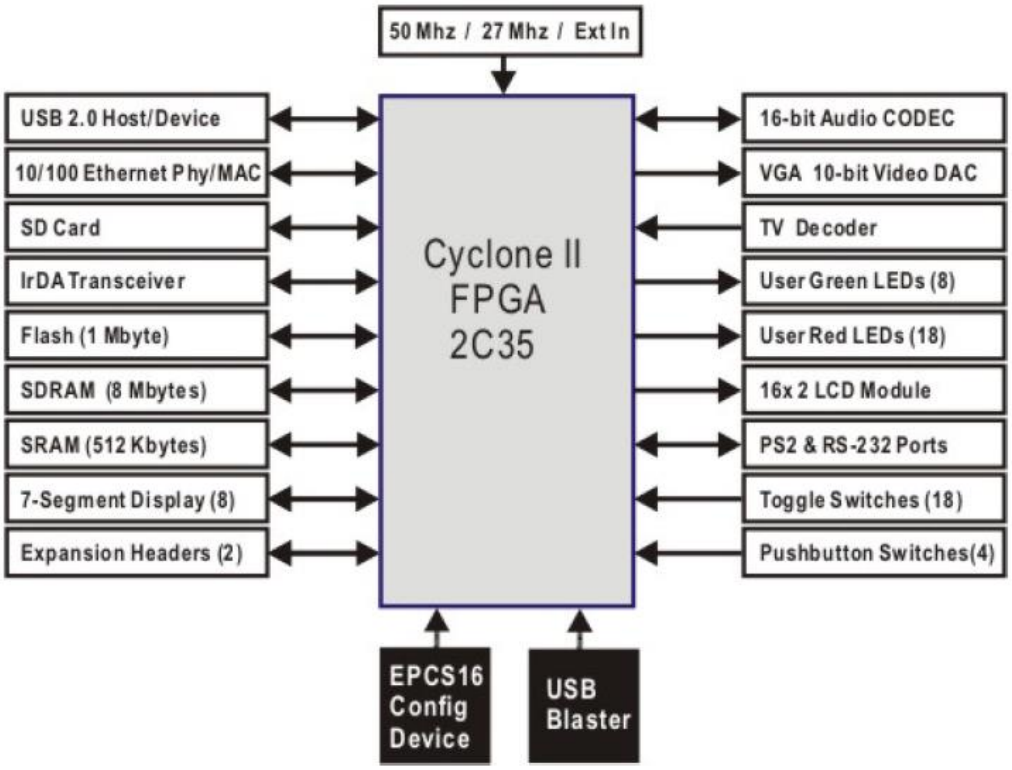


Figure 2.2: Block Diagram of DE2 Board.(Corporation 2006)

2.2 Nios II Processor

According to Gartner Research(Corporation 2012), Altera Nios II processor is said to be the world most versatile processor and the most widely used softcore processor in the history of FPGA. It is a 32-bit embedded processor architecture which is specifically designed for the Altera FPGA family. It is really flexible and ASIC-optimized. The soft-core nature of the Nios II enables the users/designers to use and generate their own custom Nios II core based on their specifications. There are 3 different categories of Nios II processors which uses Harvard architecture:

1. Nios II/e (economy)

   It has low number of logic elements which is 600. It is an ideal processor for microcontroller applications.

2. Nios II/s (standard)

   The unique feature about this processor is its real time performance is jitter free. It is also an ideal real-time processor to be used with the DSP builder-based Hardware accelerator to provide real-time high performance results. It is also supported by industry-leading Real-Time Operating Systems (RTOS). One of its features is also custom instructions, which means the ability to use FPGA for function acceleration.

3. Nios II/f (fast)

   The Nios II fast processor core can use a memory management unit (MMU) to run embedded Linux with just a simple configuration option. Both the open source and also the commercially supported versions of Linux for Nios II processors are available.

   The Avalon switch fabric is used by the Nios II processor as the interface to its embedded peripherals. The Avalon switch fabric uses a slave-

die arbitration scheme, which lets multiple masters operate simultaneously. The traditional bus on the other hand, only lets one bus master access the bus at a time. The development of Nios II can be subdivided into two parts, which are hardware generation and software creation. The hardware generation part in done using the Quartus II software through SOPC builder. The Quartus II will perform the synthesis, and place and route operation for the implementation of the entire system inside FPGA. For the software creation part, the Embedded Design Suite (EDS) will manage the software development. The EDS includes C/C++ compiler, debugger, and also an instruction set simulator.

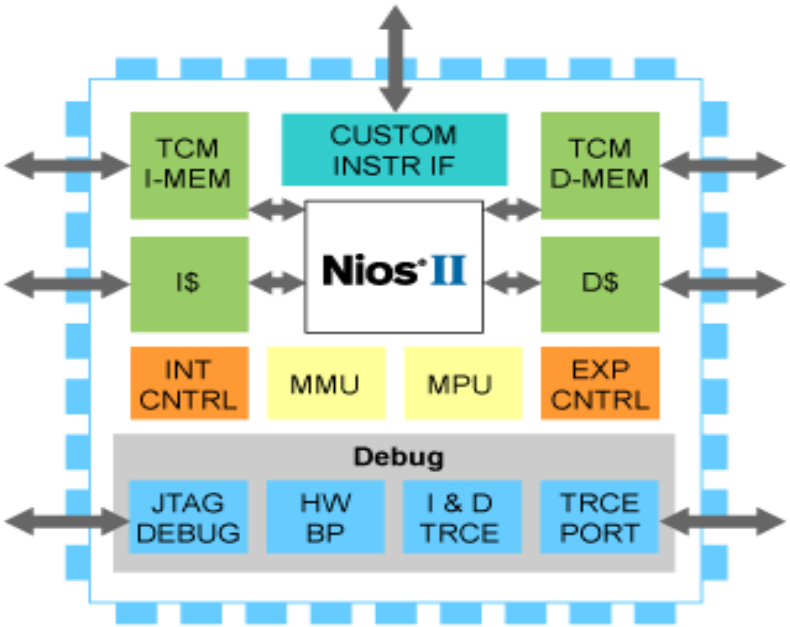


Figure 2.3 : Nios II Processor.(Corporation 2012)

In this Remote Video Surveillance Project, Nios II processor will be used for the purpose of Ethernet connectivity between FPGA and router. The code or TCP/IP core protocols will be written in Verilog in Quartus II and in C in the EDS.

2.3 TCP/IP Protocol

TCP stands for Transmission Control Protocol and IP stands for Internet Protocol. TCP/IP is actually a set of interconnected communication protocols called the TCP/IP core protocols. The internet protocol or IP is a connectionless and also unreliable protocol which has the task of addressing and packets routing between hosts. What does it mean by connectionless is that there are no session established before data exchange. It is unreliable because deliveries of packets are not guaranteed due to the probability of being lost, delivered out-of-sequence, delayed or duplicated. This IP will not make any attempt to recover any of these errors. TCP on the other hand is reliable and provides delivery service of data though established connection. The data will be transmitted in segments where each segment transmitted is assigned a specific number to achieve reliability. An acknowledgement (ACK) signal must be returned by the receiving hosts for each segment sent within a specific period of bytes received by the host. The data will be retransmitted if ACK if nor received (when there is no acknowledgement).

TCP/IP actually provides connectivity and also specifies how data should be addressed, formatted, transmitted, routed, and received at the destination. There are four abstraction layers, each having their own protocols.

1. Link Layer

It is the lowest layer which is commonly known as Ethernet. It contains communication technologies for a local network. The function of the link layer is to move packets between the Internet Layer interfaces of two different hosts that are on the same link.

2. Internet Layer(IP)

It establishes internetworking by connecting local networks. It is also responsible in sending packets across multiple networks. What it means by internetworking is sending data from a source network to a destination network. This is what called as routing. The IP performs two basic functions which are host addressing and identification, and also packet routing. This layer only provides host-to-host unreliable datagram transmission facility on different IP networks by forwarding the transport layer datagrams to the next router for relaying it further to its destination.

3. Transport Layer(TCP)

t establishes and handles host-to-host connectivity. It is responsible for end-to-end message transfer disregarding the underlying network. This message transfer includes error control, flow control, congestion control, application addressing, and segmentation. This layer can be categorized in two different category, which are connection-oriented and connectionless. The TCP falls into the connection-oriented category. This layer deals with the connections opening and maintaining between internet hosts.

4. Application Layer

This is the highest layer where applications creates user data in order to communicate this data to other processes or applications on the same or another host. This is the layer where the protocols such as SMTP, FTP, HTTP operates. The application layer actually uses the lower layers to provide a stable network connection across which to communicate.

2.4 Ethernet

Ethernet is basically a technology for local area networking (LAN). It is used to connect devices in close area. For example, connecting all computers in a building would require an Ethernet port in all the computers. For human to communicate with each other, we need a language. The same concept applies to computers that want to communicate with each other through Ethernet port. This is where TCP/IP protocol comes in, as a medium of communication or the language. Ethernet was developed in 1970's by Xerox, before becoming popular when Digital Equipment Corporation and

Figure 2.4: Cat5 Cable with RJ45 Connector.(Staff 2012)

Intel joining Xerox in developing the Ethernet standard. In 1985 it was officially accepted as IEE standard 802.3 Ethernet uses CSMA/D during packets transmitting. CSMA/D stands for Carrier Sense Multiple Access with Collision Detection and it is an algorithm for packets transmission and receiving over a common network. Type of cable used for Ethernet port connection is cat5 cable with RJ45 connector.

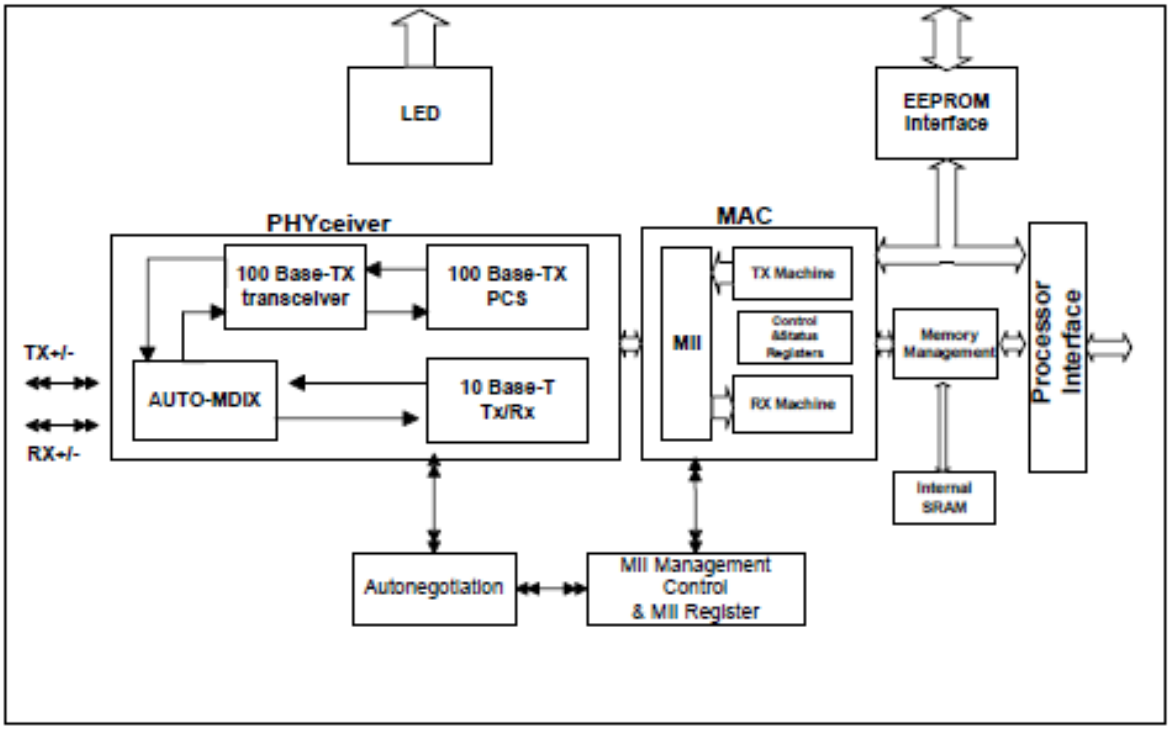


Figure 2.5: Block Diagram for DM9000A.(Inc 2006)

Altera FPGA DE2 board has 10/100 Ethernet Controller with a connector. The DE2 board uses Davicom DM9000a as the Ethernet controller, which has a general processor interface. It is a cost-effective and low pin-count single chip Fast Ethernet Controller. It has integrated MAC and PHY and supports 100base-T and 10 base-T applications. Moreover, it is also fully in accordance with the standards IEEE 802.3u specifications. It also supports IP/TCP/UDP checksum generation and checking. DM9000a supports 8-bit and 16-bit data interfaces to internal memory accesses.

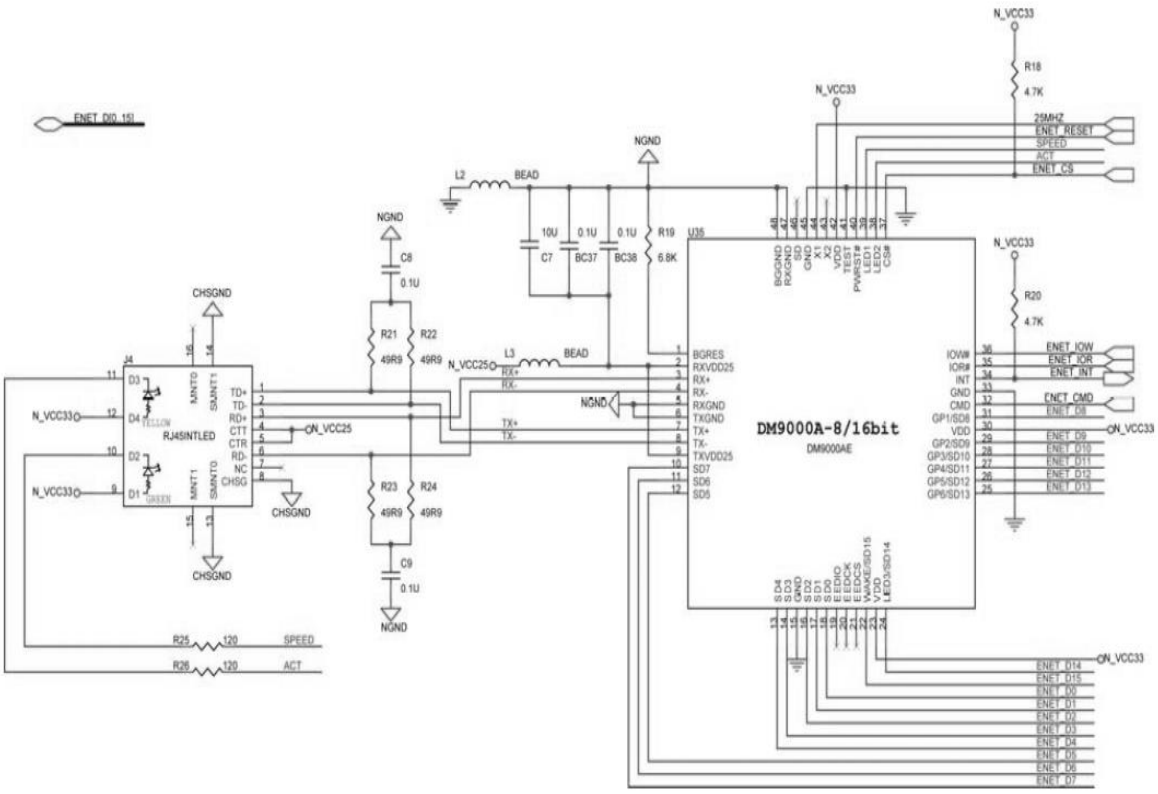


Figure 2.6: Fast Ethernet interface schematic inside FPGA.(Corporation 2006)

2.5 Universal Serial Bus (USB)

USB that was developed in the mid-1990s is an industry standard that defines cables, connectors and communications protocols that are used in bus for the purpose of data transfer, device communication, and power supply to connected devices. Nowadays, all computers have a USB connectors. With the USB connectors, many devices can be connected to the computer such as mouse, keyboard, printers, and even handphones. In 1996, USB 1.0 was introduced which has data transfer rates of 1.5Mbits/s at Low Speed and 12Mbits/s at Full Speed. USB 1.1 was released in 1998 and this was the connector that was widely used in that time. In April 2000 USB 2.0 was released. It has a higher data transfer rate with 480Mbits/s which is 40 times

better than that of USB 1.1. USB 2.0 is backward compatible with its predecessor USB 1.1. Even though USB 1.1 is the older version, it is forward compatible with USB 2.0. In 2008, USB 3.0 was introduced. It has a data transfer rate of 5 Gbit/s and it is backward compatible with USB 2.0. Compared to USB 2.0 the USB 3.0 has an increased bandwidth, where it uses two unidirectional data paths for data transmitting and data receiving instead of a one-way communications. There are two USB standard connectors which are "A" and "B". A connectors connects towards computers (head upstream), while B connectors connect to individual devices (head downstream).

Figure 2.7: Type A

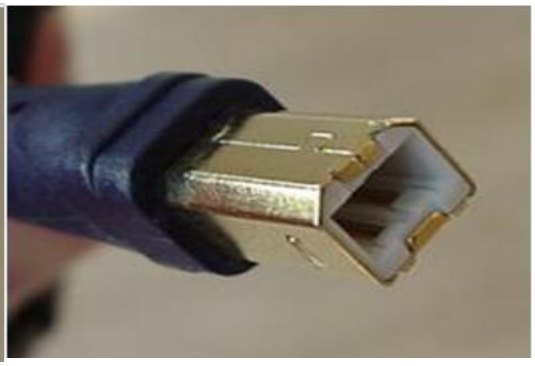


Figure 2.8: Type B(Brain)

The USB process is basically starts when the power up of the host occurs. The host will query all the devices connected to the bus and will assign an address to each one. This process is called enumeration. With the connection established, the host will find what kind of data transfers to perform which are:

1. Interrupt

   The interrupt mode will be chosen by devices like mouse or keyboard which only send very little data.
2. Bulk

   Bulk transfer mode a used by devices like printers which receives data in one big packet. Data is sent to the printer in 64-bit chunks and will be verified later to make sure its validity.
3. Isochronous

Isochronous mode is used by streaming device such as speakers where data streams between the devices in real-time and does not involve any error correction.

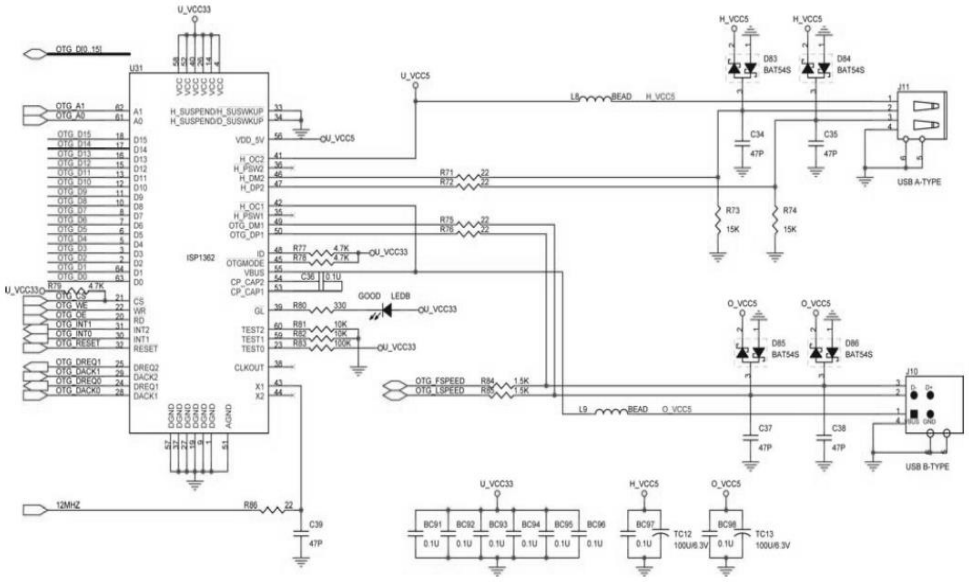


Figure 2.9: USB (ISP1362) Host and Device Schematic.(Corporation 2006)

In FPGA DE2 board, It provides USB Host/Slave controller with USB type A and type B connectors. It complies with USB 2.0, where it supports data transfer at full and low speed, and also supports both USB host and device. This USB host and device interfaces are provided using the Philips ISP1362 single-chip USB controller.

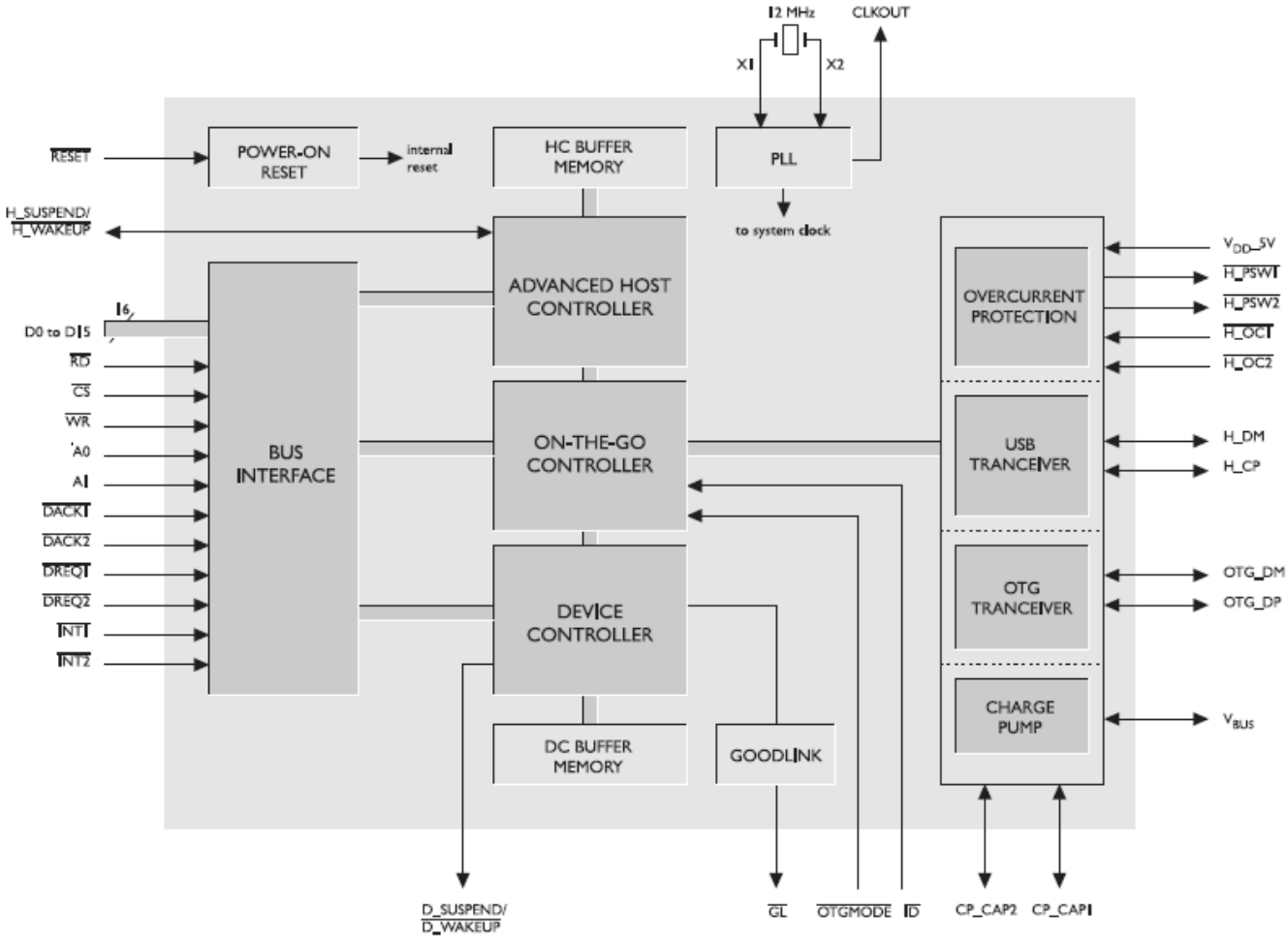


Figure 2.10: ISP1362 Architectural Diagram.(Semiconductor 2002)

The ISP 1362 includes the OTG controller, where a bus interface connects a host controller and a peripheral controller to the external CPU. Both the host controller and peripheral controller includes built-in memory to buffer USB traffic. The USB OTG controller also provides control, switching functions, and monitoring necessary for the OTG operations.

2.6 Related Works

2.6.1 The Design of Remote Image Monitoring System Based on DM9000A.

There are some previous projects done involving the DM9000a Ethernet Controller Chip. For example is The Design of Remote Image Monitoring System Based on DM9000A. (Ping Xue 2011) et al adopts the technology of SOPC hardware and software co design by taking advantage of the high speed parallel computing ability of the hardware circuit and also the flexibility in control of the software. The network card controller and the image acquisition controller uses the IP core for the system design. The control of the exposure time and the switching of the image acquisition controller are done using the Nios II software core. Instead of TCP/IP. UDP protocol is used in this project to transmit data. The system IP core can be designed usig SOPC and realized by using hardware description language (HDL). The board used in this project is DE2 board with Cyclone II EP2C35. The Terasic

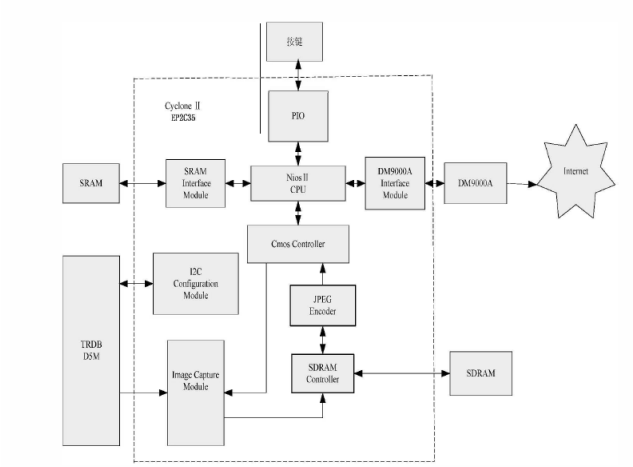


Figure 2.11: Hardware Principe Diagram (Ping Xue 2011)

TRDB_D5M digital camera development module is used as the image sensor. In the system, Nios II processor controls the CMOS controller to get the video image. For the data buffer, SDRAM is used and to compress the image, JPEG encoder is used. UDP protocol is used by the DM9000A to send the image data. The hardware circuit for this circuit is constructed in SOPC builder where the Nios II softcore is connected

with, TIMER, SRAM, JTAG, CMOS controller and DM9000A controller to form a chip programmable system.

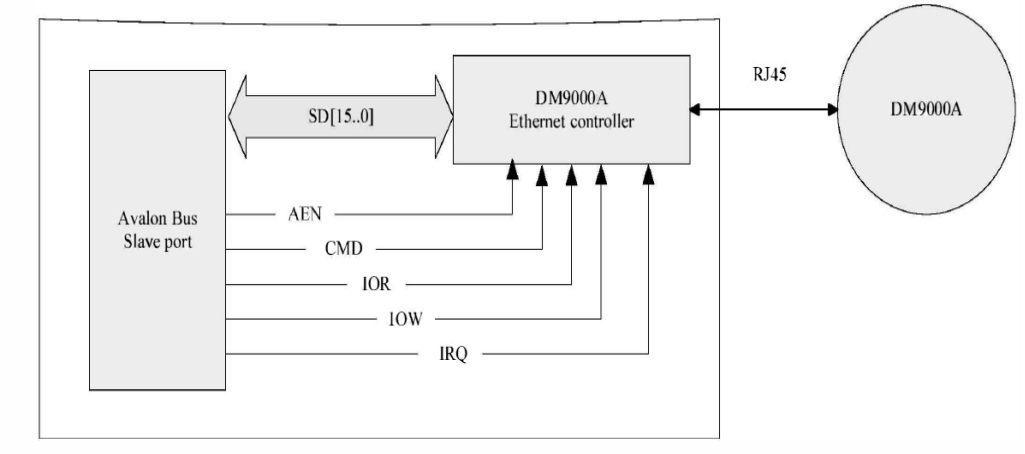


Figure 2.12: The Connection of DM9000A with Nios II CPU. (Ping Xue 2011)

2.6.2 A Reconfigurable Hardware Networking Platform for Smart Grid.

The purpose of this project is to achieve high performance and secure network communication system based on the implementation of two type of protocol stack, which are the NicheStack TCP/IP and also Open TCP/IP Hard Core(Rami Amiri 2012). For the NicheStack TCP/IP, in order to achieve a secure communication system, the ECC (Elliptic Curve Cryptography) C code is integrated into the web server source code. HTTP client is used to access the web server from the internet. The Open TCP/IP hard core was designed using VHDL. The purpose of this is to enhance the Smart Grid infrastructure. The TCP/IP core is divided into several distinctive modules. Each of the module is represented as a set of Finite State-Machine transitions. The modules that define the TCP/IP hard core are UDP, TCP, IP, ICMP, and ARP. For the TCP module, its purpose is to communicate with the application layer. The TCP module interprets applications instructions and controls other modules. Buffers and tables such as socket table are defined by the TCP module to store information

about TCP connections. The main components of the TCP are the Send and Receive module.

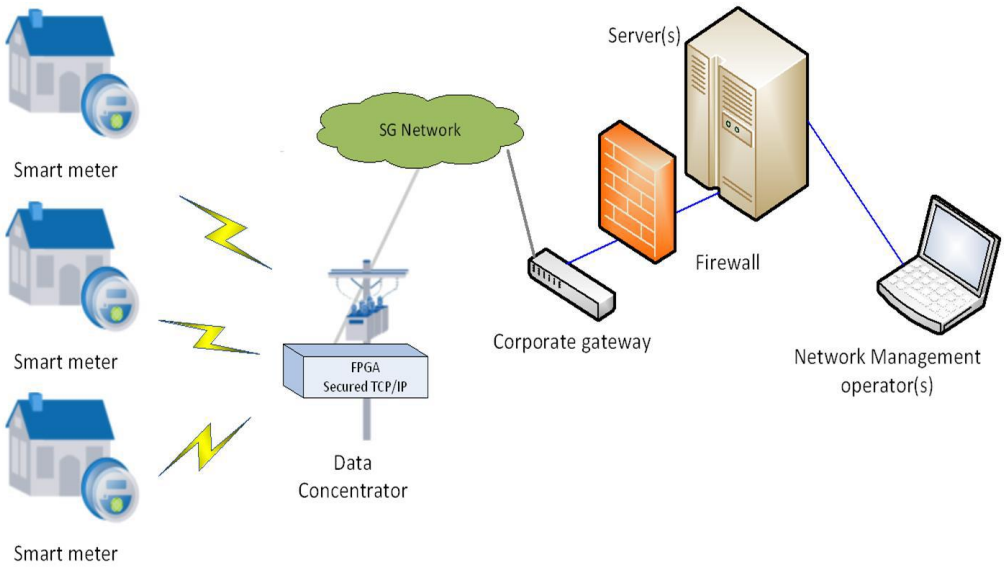


Figure 2.13: Proposed FPGA Based Data Communication for Smart Grid (Rami Amiri 2012)

The NicheStack TCP/IP was written in C and the Open TCP/IP hard core was written in VHDL. The two modules were compared in terms of speed, FPGA utilization, and SDRAM code size. The results are as follow:

Table 2.1: C Code vs VHDL Implementation

Implementation	FPGA Utilization	SDRAM code size	Speed/Latency
C Code	16%	822KB	50MHz
VHDL	12%	N/A	1000MHz

2.6.3 Secure Remote Reconfiguration of an FPGA-based Embedded System.

This project(An Braeken 2011) is about the implementation of the protocol, architecture, and implementation details of an FPGA-based embedded that is able to remotely and securely reconfigure FPGA by using TCP/IP protocol. There are mainly two blocks at the embedded system (ES) side which are one communication component for communicating over TCP/IP, and one hardware IP core for the purpose of encrypting and decrypting the communication on one hand and for authentication of entities and the data on the other hand. There are two standalone options for using TCP/IP currently exists in embedded systems which are micro IP (µIP) and light weight IP (LWIP) .It requires a lot of work to port µIP to MicroBlaze even though it is the smallest one . LWIP already exist for Xilinx FPGA, but the size is large. The C #implementations of the TCP/IP protocol are as follows

1) A synchronization package is sent by the client to the server (SYN).
2) The server accepts the SYN signal and replies with an acknowledge (SYN ACK).
3) Data transfer are initiated after the client acknowledges it too with ACK.
4) A push command is sent (PSH) after the last transmitted byte. With this, the sender accentuates that all bytes are sent and the receiver can subsequently empty its receiver buffer (PSH).

The communication will be terminated, mainly due to the triggering by (FIN) and acknowledged by reply (FIN ACK).

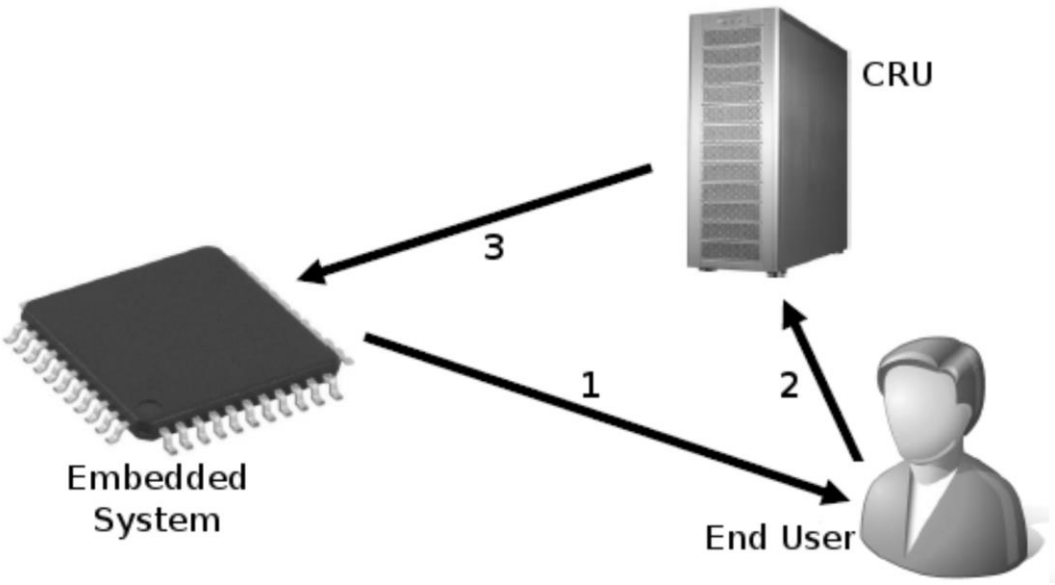


Figure 2.14: Entity Communication Channels (An Braeken 2011)

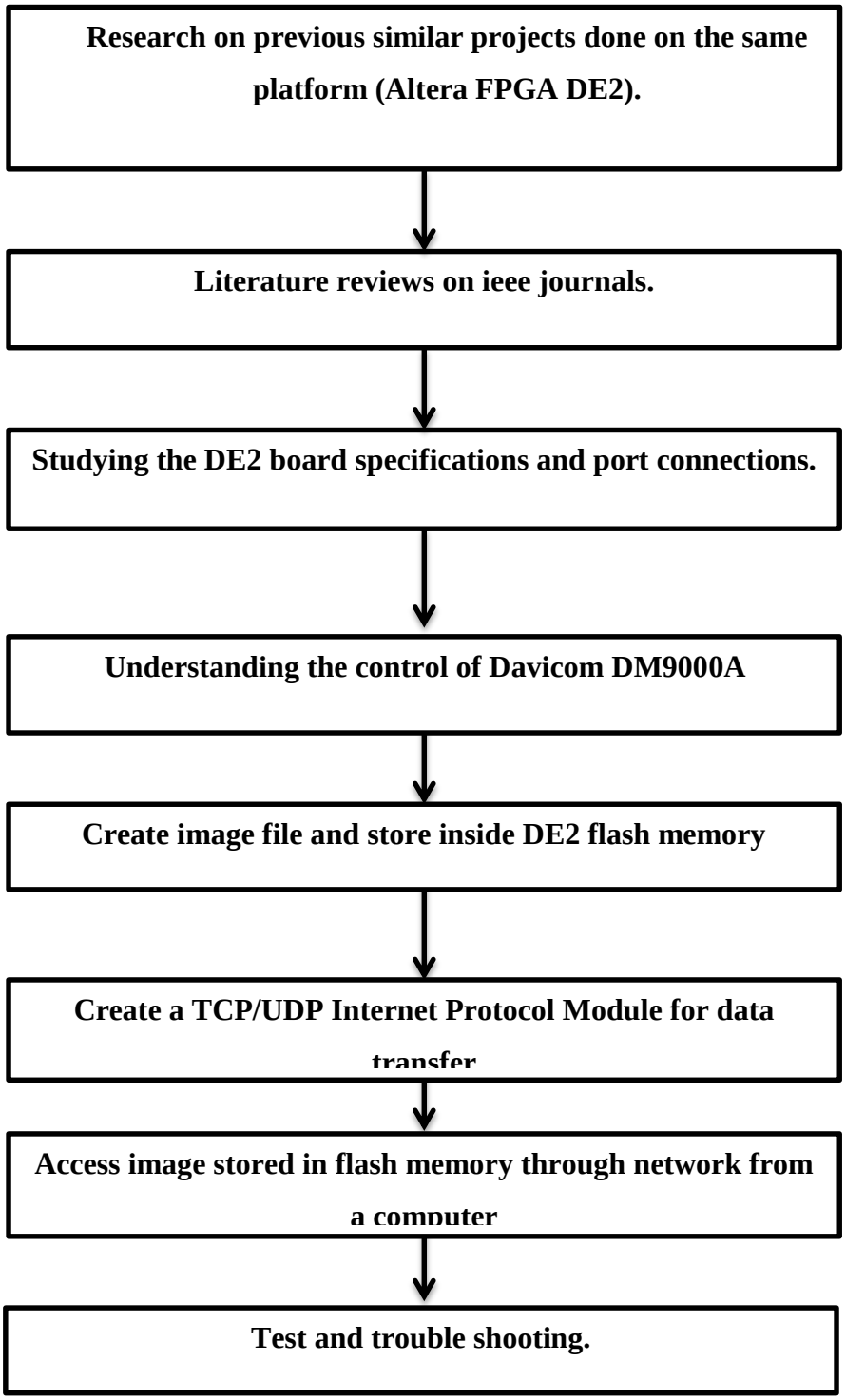
CHAPTER 3

RESEARCH METHODOLOGY

The contents covered in this chapter discuss the technique, approach, and the method used to fulfil the projects objectives and also discusses the flow of the project. A systematic and well planned approach is necessary in order to complete this project.

3.1 Work Flow

This project is divided into several steps to accomplish in order to accomplish the objectives of this project. The following flowchart shows the step by step approach to reach the objectives which starts from the research of the similar projects, the needs of this project, and the required tools until the accomplishment of the overall project objectives.


Research on previous similar projects done on the same platform (Altera FPGA DE2).
Literature reviews on ieee journals.
Studying the DE2 board specifications and port connections.
Understanding the control of Davicom DM9000A
Create image file and store inside DE2 flash memory
Create a TCP/UDP Internet Protocol Module for data transfer
Access image stored in flash memory through network from a computer
Test and trouble shooting.

3.2 Project Design

This project design consists of three parts, which are hardware design, software design and image file creation and storing.

3.2.1 Hardware Design

The hardware design of this project is done using Altera Quartus II Software, which incorporates the use of SOPC builder. SOPC builder is a software developed by Altera which is integrated in the Quartus II software. Using this SOPC components, the soft-hardware components will be created and interconnected. There are library of pre-made components. The main components that are going to be inserted using this is Nios II processor, RAM, Flash memory, Switches, LCD display, DM9000A Ethernet controller chip, and Jtag UART.

The interconnection between this components are made through Avalon bus. For the part of bus width matching, bus arbitration, and clock domain crossing, it will all be handled by the SOPC builder. Apart from using the SOPC builder to create the soft-hardware components, Verilog language will be used to create the top-level module.

3.2.2 Software Design

The software design of this project is done using the Nios II IDE software, which is part of the Altera software package that comes with the Quartus II software. The software part is actually a coding which are done using C language. There are several coding need to be done in order to create a complete system. These includes the creation of Internet protocol (IP) for transport layer which is either TCP or UDP, the creation of c file for retrieving the image from the flash memory, creating c files for the initialization and setting of IP address, and MAC address, creating c file for LCD display, and the creation of c file for the protocol and communication of the DM9000A Ethernet controller chip.

3.2.3 Image File Creation and Storing

The image file will be created in a format so that it can be stored inside the flash memory of FPGA DE2. In order for the image to be viewed through the browser, it will be stored in an html format where the image will be embedded in it.

3.3 TCP vs UDP

TCP stands for transmission control protocol and UDP stands for user datagram protocol. TCP is a connection-based protocol which means it is reliable and ordered. When a message/data/packet is sent, if there are no connection fails, it will get delivered. If loss of connection happens, the lost

part will be requested by the server. So, there can be no corruption while message transferring occurs. Unlike TCP, UDP protocol is connectionless. Whenever a message is sent, there will be no guarantee that the message is received by the receiver. The sent message could also been corrupted during sending. So, this UDP is unreliable. Furthermore, TCP is ordered. The data sent and the received data are in the same order. For UDP, the received data may not be the same order as the sent data. Moreover, TCP is less susceptible to attack. When packets/data sent from sender to receiver, there will be acknowledgment sent from the sender to receiver and also acknowledgement from receiver to sender to confirm data has arrived. So, it is hard for attackers/hackers to modify or delete any data. For UDP, there is no acknowledgement from both side of sender and receiver, which means it has no built-in mechanism to guarantee delivery of data and also retransmit of lost data. So, it is easy for attackers/hackers to insert, delete, or modify the data/packets sent. Due to important of reliable and secure data in video surveillance, TCP protocol has been chosen.

3.4 Block Diagram of Design

This is the design of the remote video surveillance system for this project in block diagram. The Nios II processor is central or core of this system. It will be connected to the flash memory and also the DM9000A Ethernet controller chip. This three components will be built using SOPC builder in FPGA. The FPGA DE2 board will be connected to the computer and the data transfer will be done through the Davicom DM9000A chip. Physically, the communication will be done using and Ethernet cable and the connection is through the RJ45 port on both the FPGA DE2 and also the computer. A router in an optional in this project, since the establishment of connection and sending data from DE2 board to computer is the main target of the project.

This project design emphasize the client server relationship. Where the computer is the client and the FPGA DE2 is the server. The client will make request to access the image file in the flash memory through a web browser, and the server will respond by executing the request.

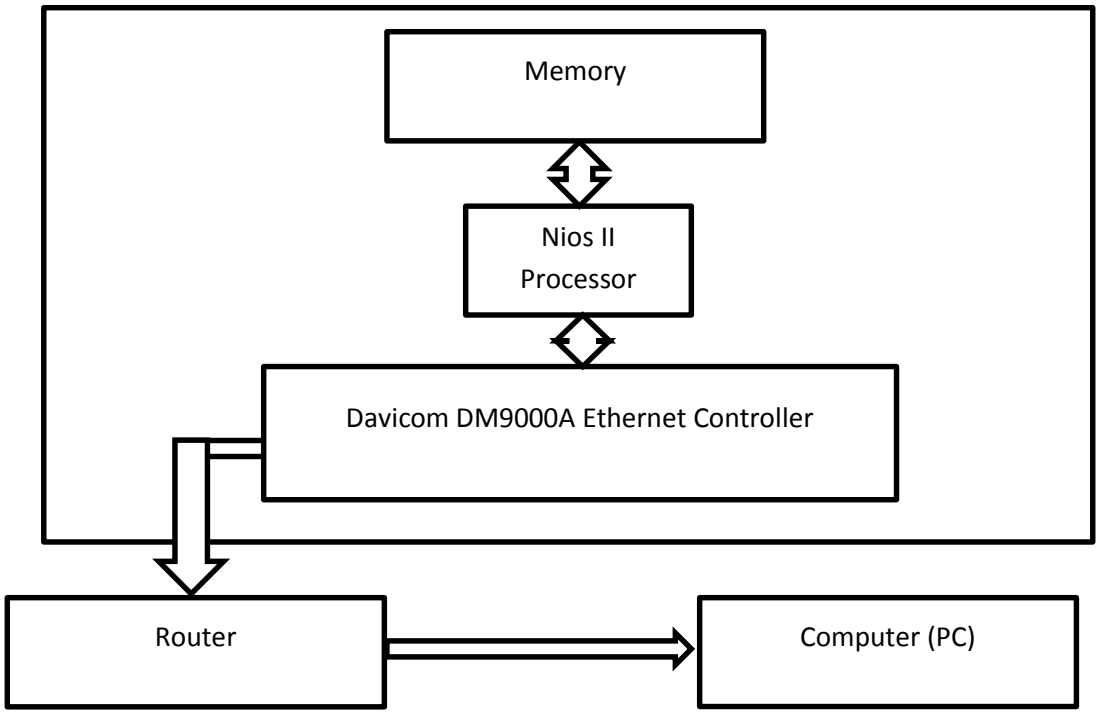


Figure 3.1: Block Design of System

3.5 Operation Flow

Figure 3.2 below shows the operational flow of the overall project, which is from the storing of image file until the display of image file on browser through the computer.

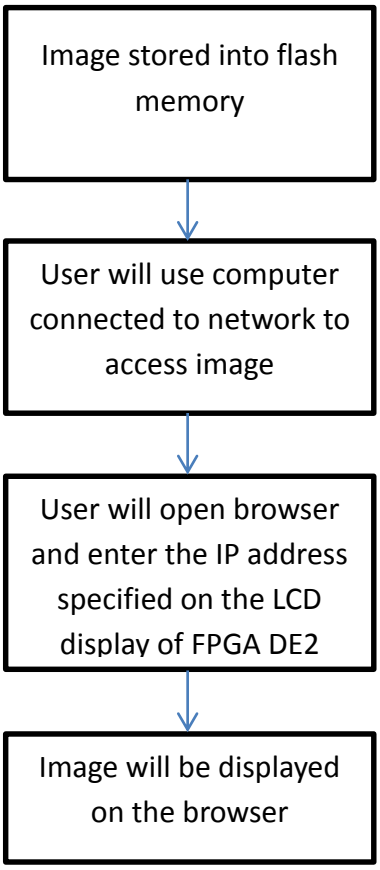


Image stored into flash memory
User will use computer connected to network to access image
User will open browser and enter the IP address specified on the LCD display of FPGA DE2
Image will be displayed on the browser

Figure 3.2: Operation Flow

CHAPTER 4

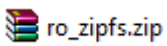
RESULTS AND DISCUSSIONS

4.1 Image File

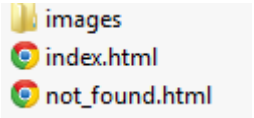
The image file is created in the format so that a computer (client) can access the image stored inside FPGA DE2 flash memory. The image file is stored in a zip format

4.1.1 Image File Creation

The image file created and stored in a zip format



The contents of this zip file are:



The folder images contain the image to be viewed. The index.html and not_found.html contents are as follows:

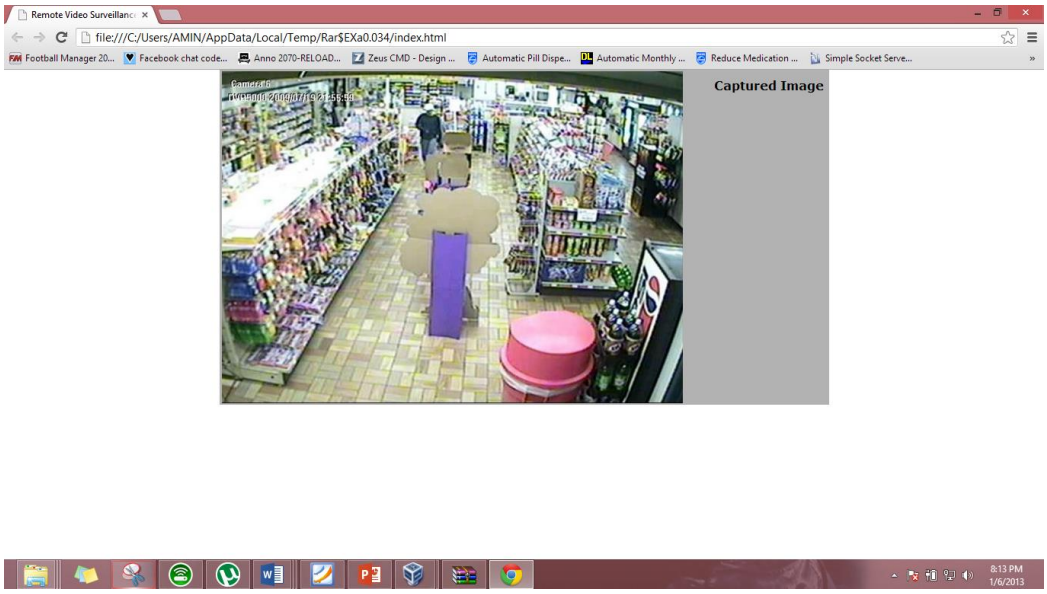


Figure 4.1: index.html

The index.html file contains the image to be viewed, when the computer access the flash memory to view this image, this file will be shown.

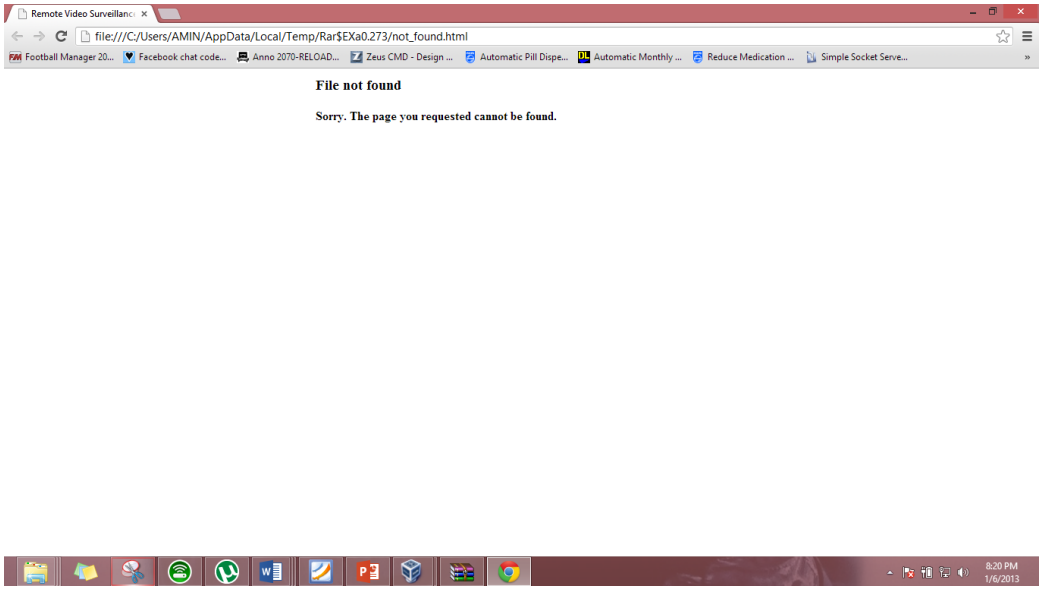


Figure 4.2: not_found.html

If the image failed to be accessed, this will be displayed.

4.1.2 Image File Storing

After the image file is created, it is stored inside the FPGA DE2 flash memory using the DE2 Control Panel.

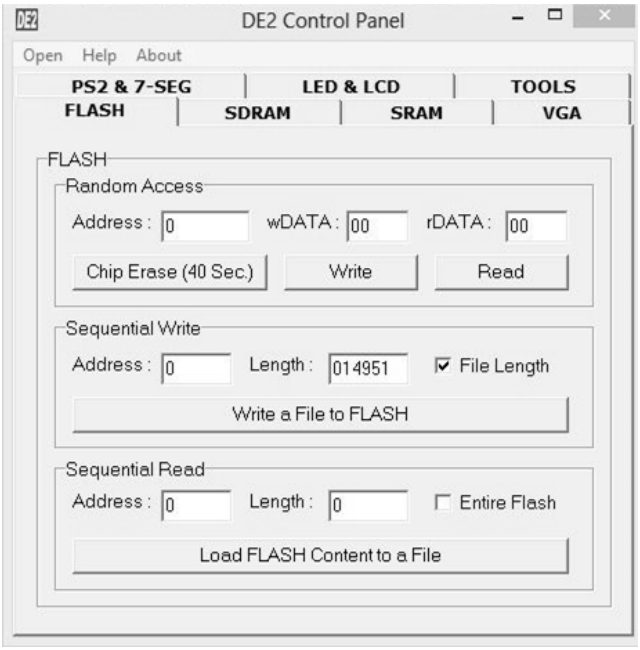


Figure 4.3: DE2 Control Panel

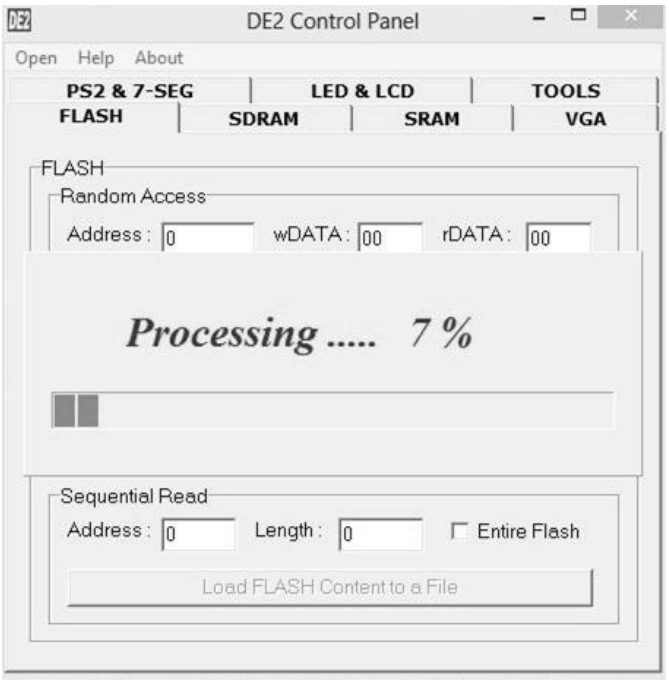


Figure 4.4: Flash Memory Erasing

Before the zip file is stored, the contents of the flash must be erased first.

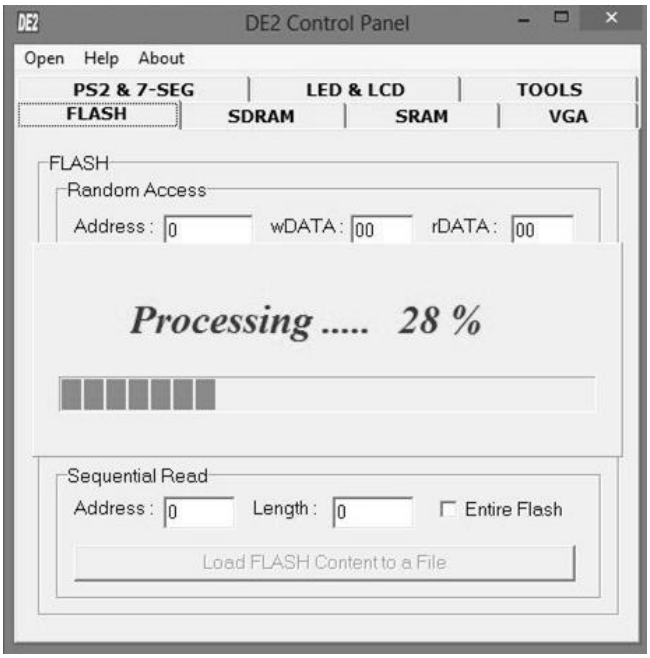
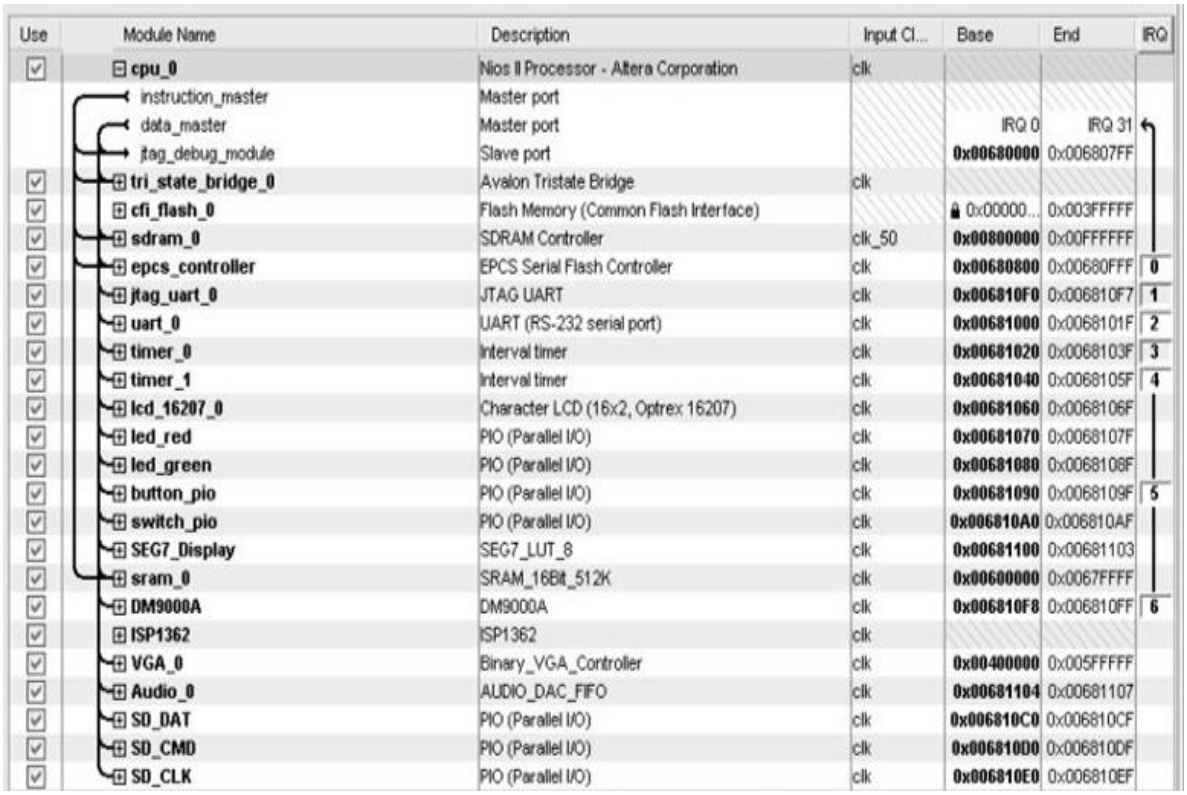


Figure 4.5: Write to Flash Memory

After, the contents has been erased, the zip file is programmed into the flash memory.

4.2 Hardware

The design of the soft-hardware part which is done using SOPC builder is as follows:



Use	Module Name	Description	Input Cl...	Base	End	IRQ
☑	⊟ cpu_0	Nios II Processor - Altera Corporation	clk			
	instruction_master	Master port				
	data_master	Master port		IRQ 0	IRQ 31	
	jtag_debug_module	Slave port		0x00680000	0x006807FF	
☑	⊞ tri_state_bridge_0	Avalon Tristate Bridge	clk			
☑	⊞ cfi_flash_0	Flash Memory (Common Flash Interface)		0x00000...	0x003FFFFF	
☑	⊞ sdram_0	SDRAM Controller	clk_50	0x00800000	0x00FFFFFF	
☑	⊞ epcs_controller	EPCS Serial Flash Controller	clk	0x00680800	0x00680FFF	0
☑	⊞ jtag_uart_0	JTAG UART	clk	0x006810F0	0x006810F7	1
☑	⊞ uart_0	UART (RS-232 serial port)	clk	0x00681000	0x0068101F	2
☑	⊞ timer_0	Interval timer	clk	0x00681020	0x0068103F	3
☑	⊞ timer_1	Interval timer	clk	0x00681040	0x0068105F	4
☑	⊞ lcd_16207_0	Character LCD (16x2, Optrex 16207)	clk	0x00681060	0x0068106F	
☑	⊞ led_red	PIO (Parallel I/O)	clk	0x00681070	0x0068107F	
☑	⊞ led_green	PIO (Parallel I/O)	clk	0x00681080	0x0068108F	
☑	⊞ button_pio	PIO (Parallel I/O)	clk	0x00681090	0x0068109F	5
☑	⊞ switch_pio	PIO (Parallel I/O)	clk	0x006810A0	0x006810AF	
☑	⊞ SEG7_Display	SEG7_LUT_8	clk	0x00681100	0x00681103	
☑	⊞ sram_0	SRAM_16Bit_512K	clk	0x00600000	0x0067FFFF	
☑	⊞ DM9000A	DM9000A	clk	0x006810F8	0x006810FF	6
☑	⊞ ISP1362	ISP1362	clk			
☑	⊞ VGA_0	Binary_VGA_Controller	clk	0x00400000	0x005FFFFF	
☑	⊞ Audio_0	AUDIO_DAC_FIFO	clk	0x00681104	0x00681107	
☑	⊞ SD_DAT	PIO (Parallel I/O)	clk	0x006810C0	0x006810CF	
☑	⊞ SD_CMD	PIO (Parallel I/O)	clk	0x006810D0	0x006810DF	
☑	⊞ SD_CLK	PIO (Parallel I/O)	clk	0x006810E0	0x006810EF	

Figure 4.6: SOPC Hardware Design

The hardware design and interconnections are as follows. Most of the components are pre-made components which are taken from the library. Even though the components are pre-made, there are settings need to be done to match the project requirement. The cpu setting is as follows:

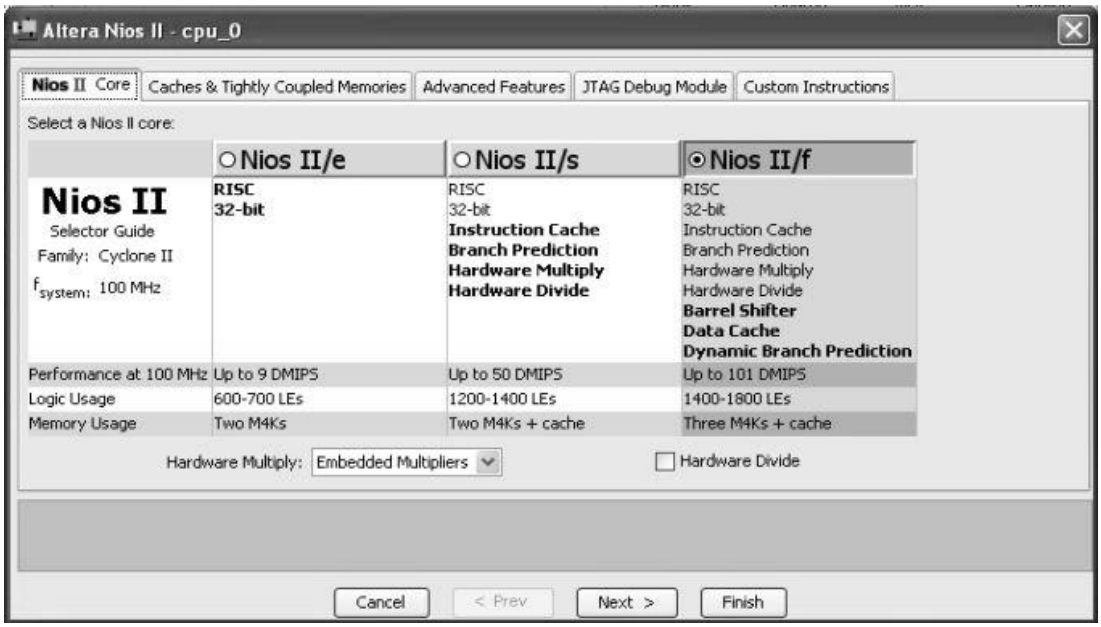


Figure 4.7: Nios II Processor Type

Nios II/f is selected where the f stands for fast. The other two types are Nios II/e for economic, and Nios II/s for standard.

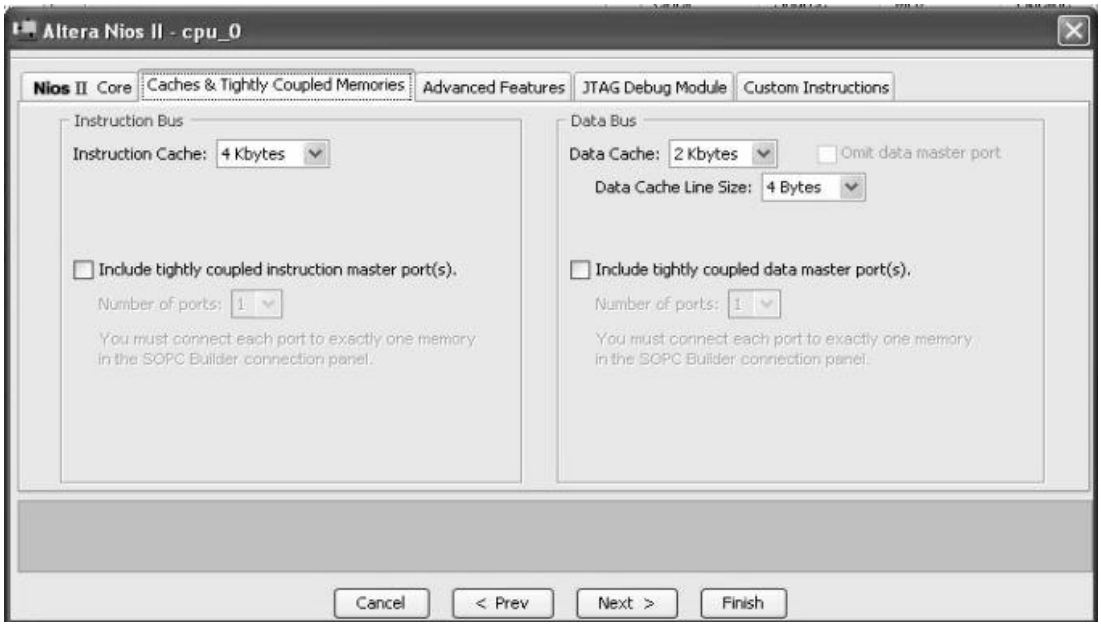

Altera Nios II - cpu_0
Nios II Core
Caches & Tightly Coupled Memories
Advanced Features
JTAG Debug Module
Custom Instructions
Instruction Bus
Instruction Cache: 4 Kbytes
Include tightly coupled instruction master port(s).
Number of ports: 1
You must connect each port to exactly one memory in the SOPC Builder connection panel.
Data Bus
Data Cache: 2 Kbytes
Omit data master port
Data Cache Line Size: 4 Bytes
Include tightly coupled data master port(s).
Number of ports: 1
You must connect each port to exactly one memory in the SOPC Builder connection panel.
Cancel
< Prev
Next >
Finish

Figure 4.8: Nios II Processor Instruction and Data Bus Setting

This window of setting shows the instruction cache for instruction bus and data cache for data bus.

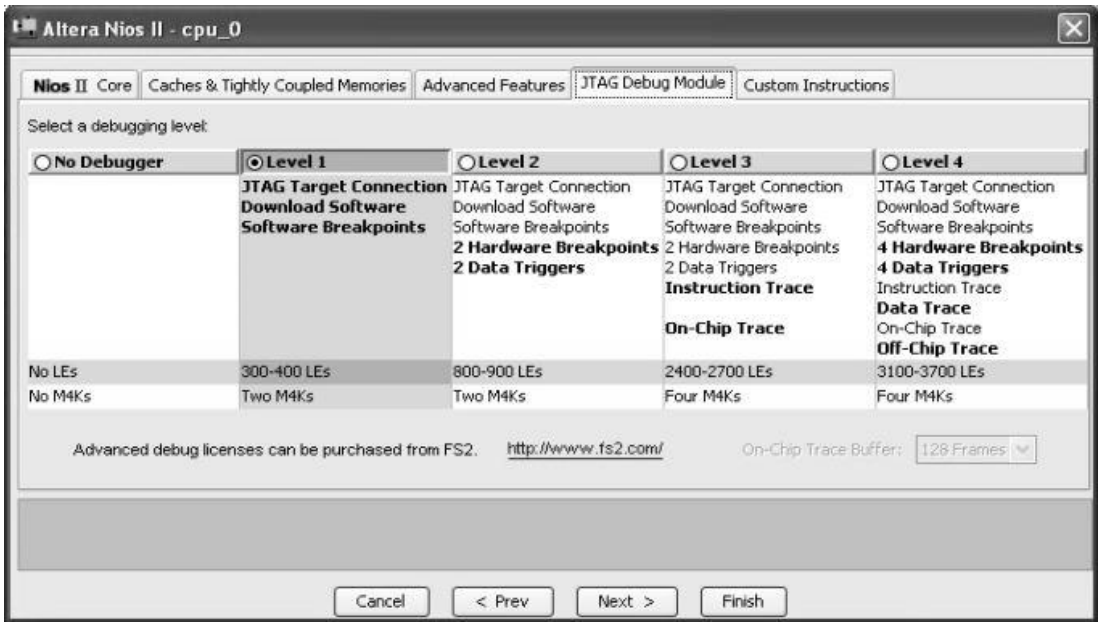


Figure 4.9: Nios II JTAG Setting

This is set as default.

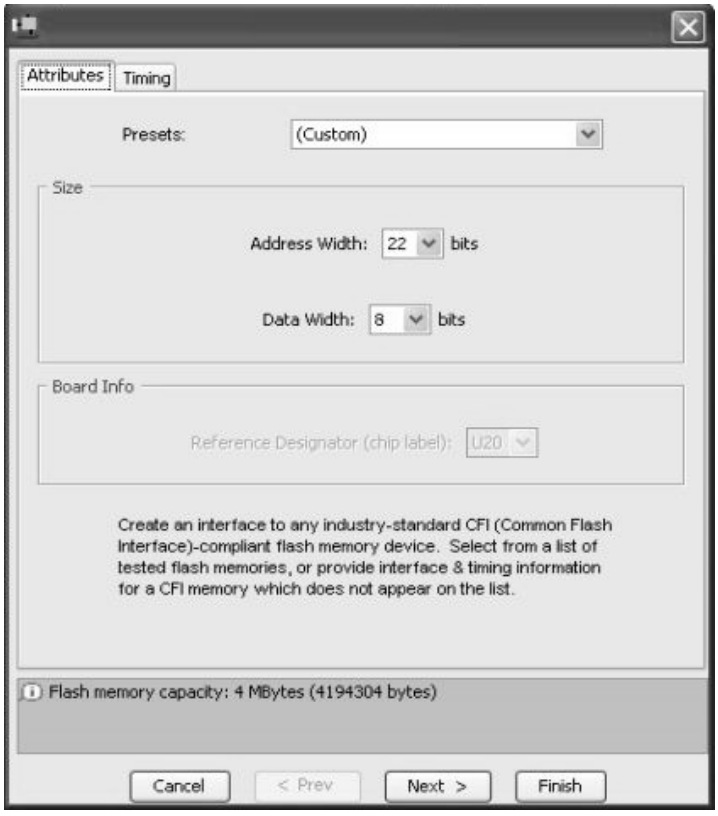


Figure 4.10: Flash Memory Data and Address Width

The flash memory address width and data width settings are as follows.

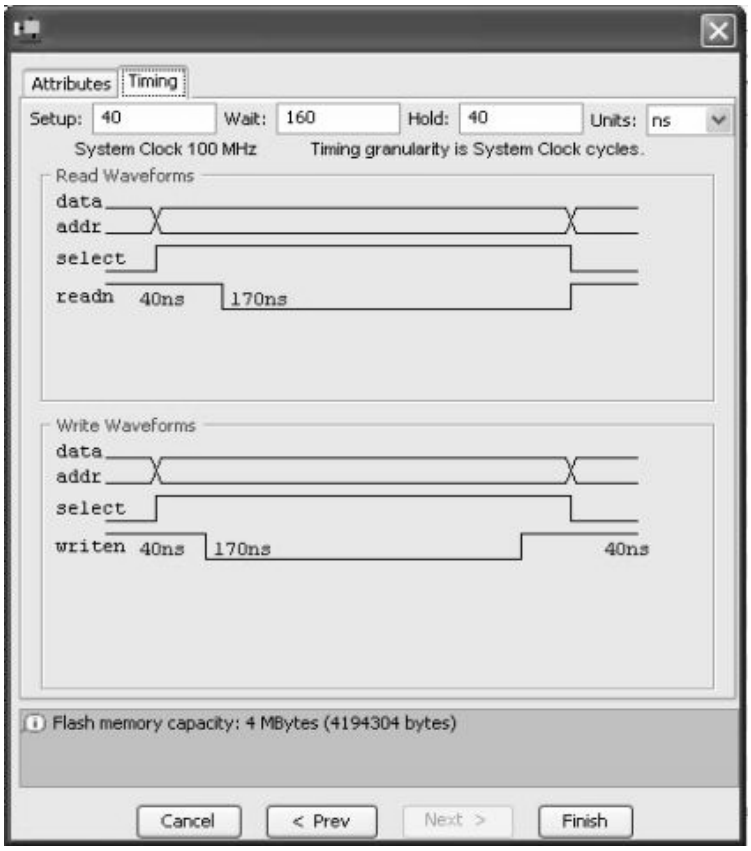


Figure 4.11: Flash Memory Timing

The timing settings which includes setup time, wait time, and hold time are as shown above. Other settings of LEDs, buttons, and switches are as follows:

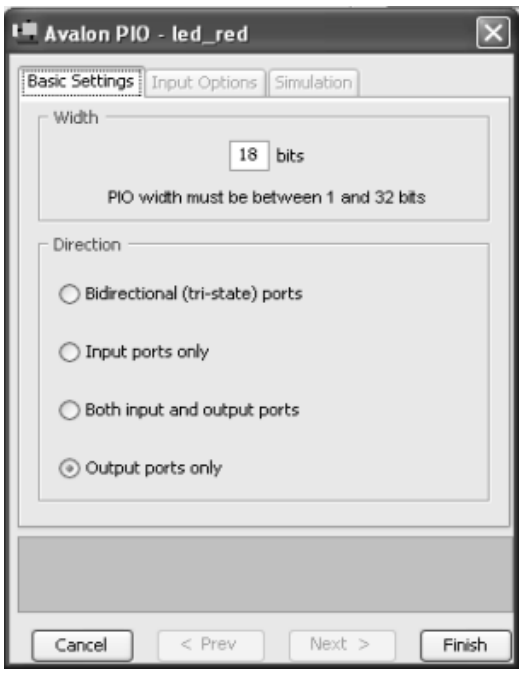


Figure 4.12: Red LED Setting

Figure 4.13: Green LED Setting

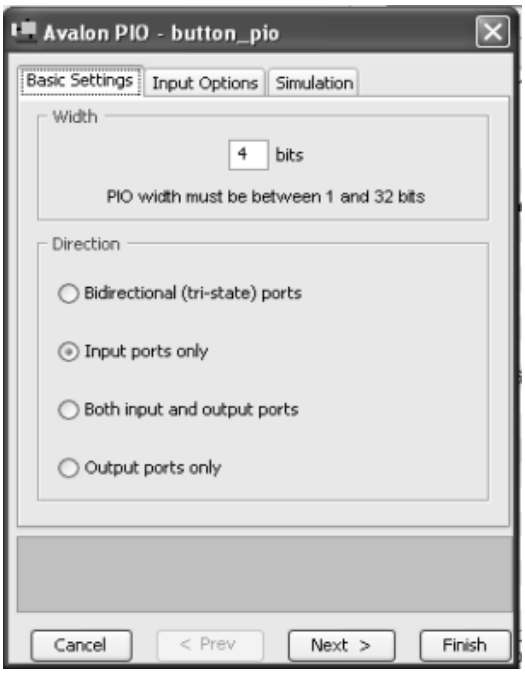


Figure 4.14: Button Setting

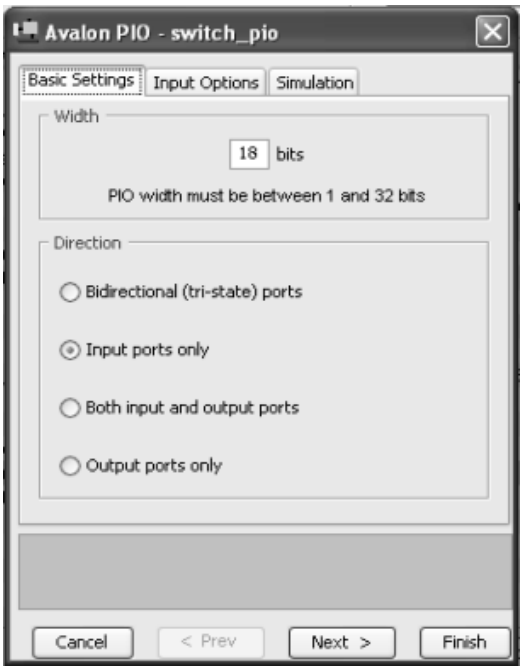


Figure 4.15: Switch Setting

There are some components which are not included in the library. These components are DM9000A and ISP1362. The DM9000A is an Ethernet controller chip while the ISP1362 is a single chip USB controller. The ISP1362 will not be used in this project but it is included in the hardware interconnection for future project improvisations. In order to create these components, a custom component must be created, and the Verilog source file which defines the interconnection of pins (the inputs and outputs) must be added into the new custom component creation menu. After that, the signal and interfaces must be set, but usually and in this case, the settings are done in default.

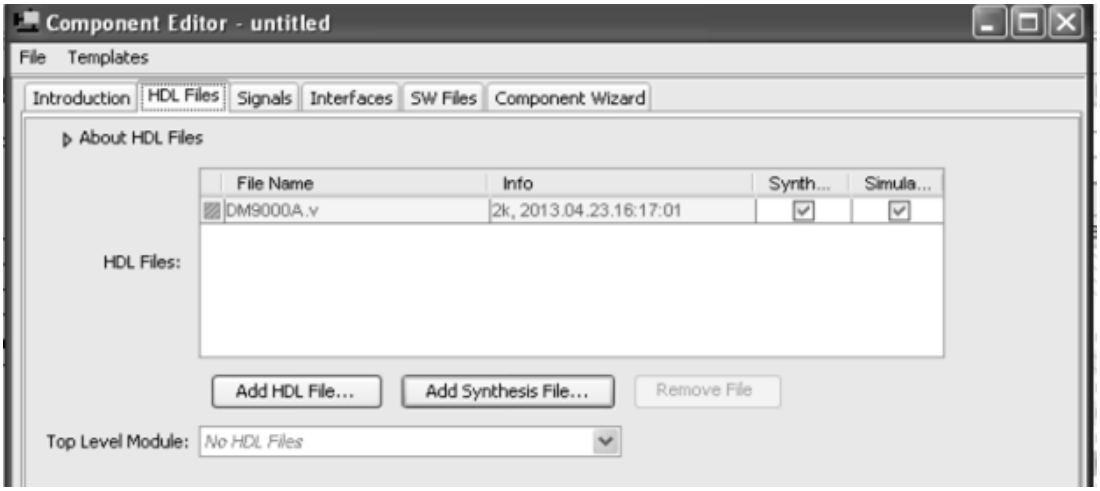


Figure 4.16: DM9000A Custom Component Creation

4.3 Software

The software has been coded and created which consists of several main parts.

4.3.1 Transport Layer Internet Protocol (IP)

A network protocol analysis has being done on a pre-made IP core which are provided by TERASIC in order to get more understanding on the protocols involving network connections. This analysis is being done using Wireshark, which is a network analyser/ packet sniffer software.

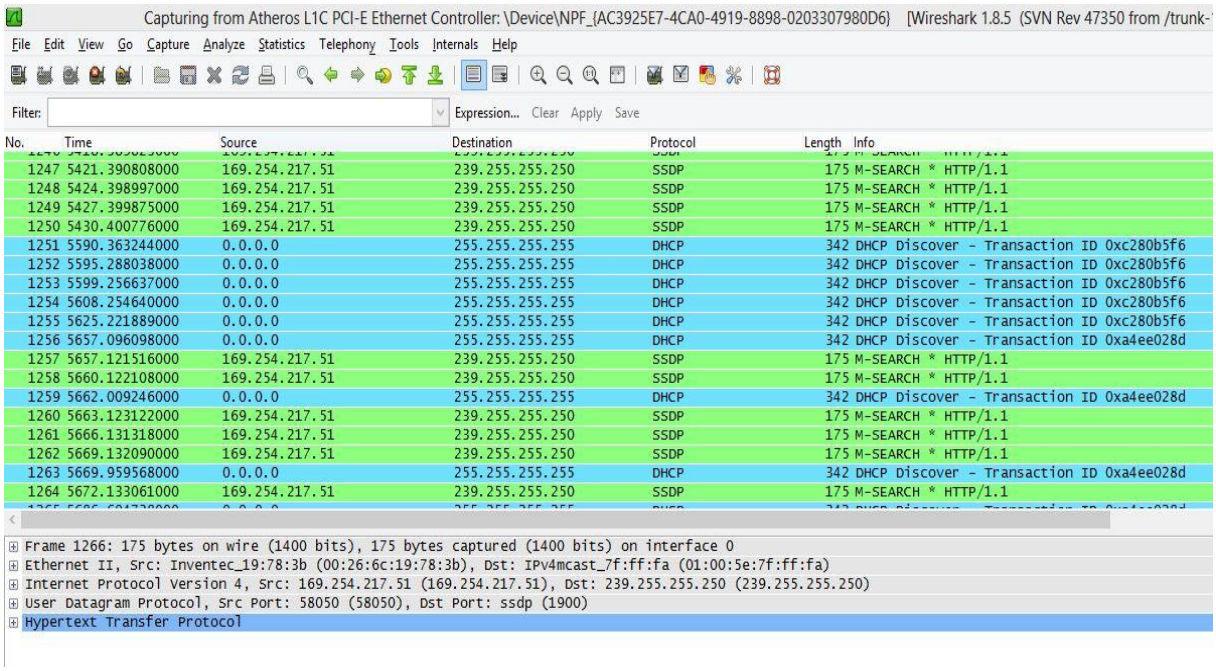
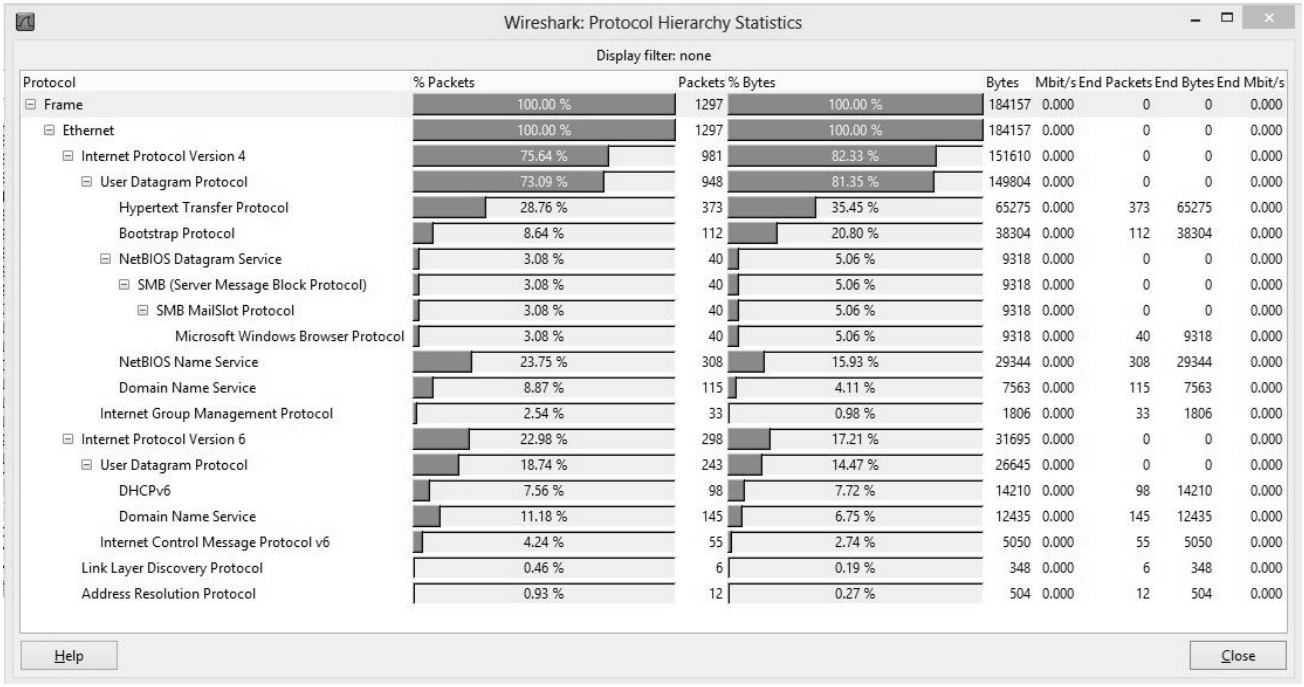


Figure 4.17: Wireshark Protocol Analysis

After the IP core is being run in Nios II, Wireshark was used to capture the interfaces after the connection has been made between the DE2 and computer through Ethernet. The above figure shows the protocol involved during the initialization of the protocol. SSDP stands for Simple Service Discovery Protocol. It is a network protocol based on Internet Protocol Suite that is used for discovery of network services. It accomplishes this without the assistance of DHCP and DNS. A client, which in this case is the computer that wishes to discover available services on a network will use the M-SEARCH method.

DHCP is an acronym for Dynamic Host Configuration Protocol. It is a network protocol that is used to configure devices that are connected to the network so the devices can communicate with each other using internet protocol (IP). It involves client and server. In this case, FPGA is the server and computer is the client.



Protocol	% Packets	Packets	% Bytes	Bytes	Mbit/s	End Packets	End Bytes	End Mbit/s
Frame	100.00 %	1297	100.00 %	184157	0.000	0	0	0.000
Ethernet	100.00 %	1297	100.00 %	184157	0.000	0	0	0.000
Internet Protocol Version 4	75.64 %	981	82.33 %	151610	0.000	0	0	0.000
User Datagram Protocol	73.09 %	948	81.35 %	149804	0.000	0	0	0.000
Hypertext Transfer Protocol	28.76 %	373	35.45 %	65275	0.000	373	65275	0.000
Bootstrap Protocol	8.64 %	112	20.80 %	38304	0.000	112	38304	0.000
NetBIOS Datagram Service	3.08 %	40	5.06 %	9318	0.000	0	0	0.000
SMB (Server Message Block Protocol)	3.08 %	40	5.06 %	9318	0.000	0	0	0.000
SMB MailSlot Protocol	3.08 %	40	5.06 %	9318	0.000	0	0	0.000
Microsoft Windows Browser Protocol	3.08 %	40	5.06 %	9318	0.000	40	9318	0.000
NetBIOS Name Service	23.75 %	308	15.93 %	29344	0.000	308	29344	0.000
Domain Name Service	8.87 %	115	4.11 %	7563	0.000	115	7563	0.000
Internet Group Management Protocol	2.54 %	33	0.98 %	1806	0.000	33	1806	0.000
Internet Protocol Version 6	22.98 %	298	17.21 %	31695	0.000	0	0	0.000
User Datagram Protocol	18.74 %	243	14.47 %	26645	0.000	0	0	0.000
DHCPv6	7.56 %	98	7.72 %	14210	0.000	98	14210	0.000
Domain Name Service	11.18 %	145	6.75 %	12435	0.000	145	12435	0.000
Internet Control Message Protocol v6	4.24 %	55	2.74 %	5050	0.000	55	5050	0.000
Link Layer Discovery Protocol	0.46 %	6	0.19 %	348	0.000	6	348	0.000
Address Resolution Protocol	0.93 %	12	0.27 %	504	0.000	12	504	0.000

Figure 4.18: Wireshark Protocol Hierarchy Statistics

The figure above shows the protocol hierarchy analysis using Wireshark. This IP core uses UDP protocol, under the UDP protocol layer, there are a lot of protocols involved, which are Hypertext Transfer Protocol (HTTP), Bootstrap protocol, NetBIOS Datagram Service, Server Message Block (SMB) protocol and SMB MailSlot Protocol, Microsoft Windows Browser Protocol, NetBIOS Name Service, Domain Name Service (DNS), and Internet Group Management Protocol (IGMP).
Each protocol has its own function. HTTP is an application layer network protocol. HTTP functions as a request-response protocol in the client-server computing model. A web browser, for example, may be the client and an application running on a computer hosting a web site may be the server. The client submits an HTTP request message to the server. The server, which provides resources such as HTML files and other content, or performs other functions on behalf of the client, returns a response message to the client.

In computer networking, the Bootstrap Protocol, or BOOTP, is a network protocol used by a network client to obtain an IP address from a configuration server. In NetBIOS Datagram Service the application is responsible for error detection and

recovery. NetBIOS Name Service is a service providing name lookup and registration. It serves the same purpose as DNS, which is to translate human readable names to IP address. It is encapsulated by UDP.

DNS translates domain names to numerical IP addresses needed for the purpose of locating computer services and devices worldwide. The Internet Group Management Protocol (IGMP) is a communications protocol which are used by hosts and adjacent routers on IP networks so that multicast group memberships can be established. IGMP is an integral part of IP multicast.

Table 4.1: TCP vs UDP

PROTOCOL	
TCP	**UDP**
Connection oriented – reliable and data not lost	Connectionless protocol - unreliable and data will be loss.
Ordered	Not ordered
Less susceptible to hack/attack	More susceptible to hack/attack

The table above is the comparison between UDP and TCP protocol. Since this project is on video surveillance, so TCP will be chosen as it is a safer and more reliable protocol.

4.3.2 The Nios II Coding

The coding of this project has been divided into 5 main parts which are web_server.c, http.c, network_utilities.c, dm9000a.c, and lcd.c.
web_server.c file contains the main function and the LWIP callback function. The LWIP callback installs task once LWIP has been properly initialized. http.c file is the implementation of an HTTP server which includes all the necessary sockets calls to

handle a multiple connections and parsing basic HTTP commands to handle GET and POST requests. HTTP GET request the image file stored inside the flash memory, where the server will fetch the file and send it to the client that is requesting the file. It also includes socket creation.

network_utilities.c contains MAC address, IP address, and DHCP routines to manage the addressing. These are implementation specific and used by LWIP during initialization. dm9000a.c defines the communication for DM9000A Ethernet controller chip. The lcd.c file is a sub-function which defines LCD initialization and how to write text/string on LCD display.

LWIP stands for Light Weight Internet Protocol. It is a small independent implementation of TCP/IP protocol. The focus of the LWIP TCP/IP implementation is to reduce the RAM usage while still having a full scale TCP. There are also other protocol stack which is Interniche stack. LWIP is specific for TCP/IP protocol only. Interniche supports other protocols, but it requires more RAM and ROM usage compared to LWIP.

4.3.2.1 IP Address and MAC Address Setting

The IP Address and MAC address can be set. The coding for setting the IP and MAC address is done in network_utilities.c. There are two possibilities of assigning the IP and MAC address, either using the switches or let DHCP assign the IP address.

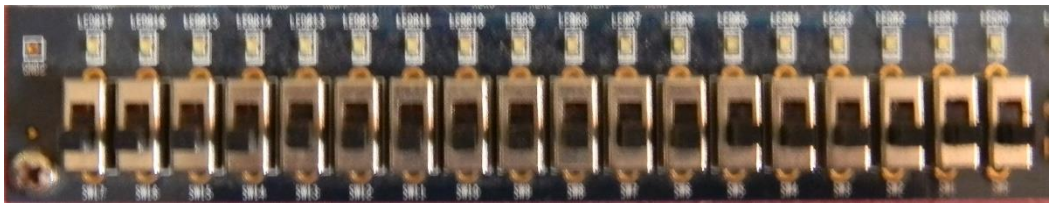


Figure 4.19: DE2 Switches

There are 18 switches in DE2 board, SW0 until SW17. SW17 used to control how the IP address is set. There are 4 segments in an IP address. The first two is set to default value and the 3rd and 4th segment is set based on switch. If the SW17 is '1', the IP address are as follows:

192.168.SW[15:12].SW[11:8]+128

If SW17 is '0', it will get the IP address from DHCP server. There are 5 segments for MAC address setting, where the first 4 segments are set to default value. The 5th segment value are set based on switches. The MAC address setting is as follows:

00-09-00-AE-SW[7:0]

4.4 Expected Result

The expected result of this project is a creation of a web server where the FPGA DE2 is the server and the computer is the client. The client will open a browser and type the IP address displayed on LCD to request access to the image from the server.

Figure 4.20: LCD Display of MAC and IP Address

The server will process the request and fetch the image file from the flash memory for the client to view.

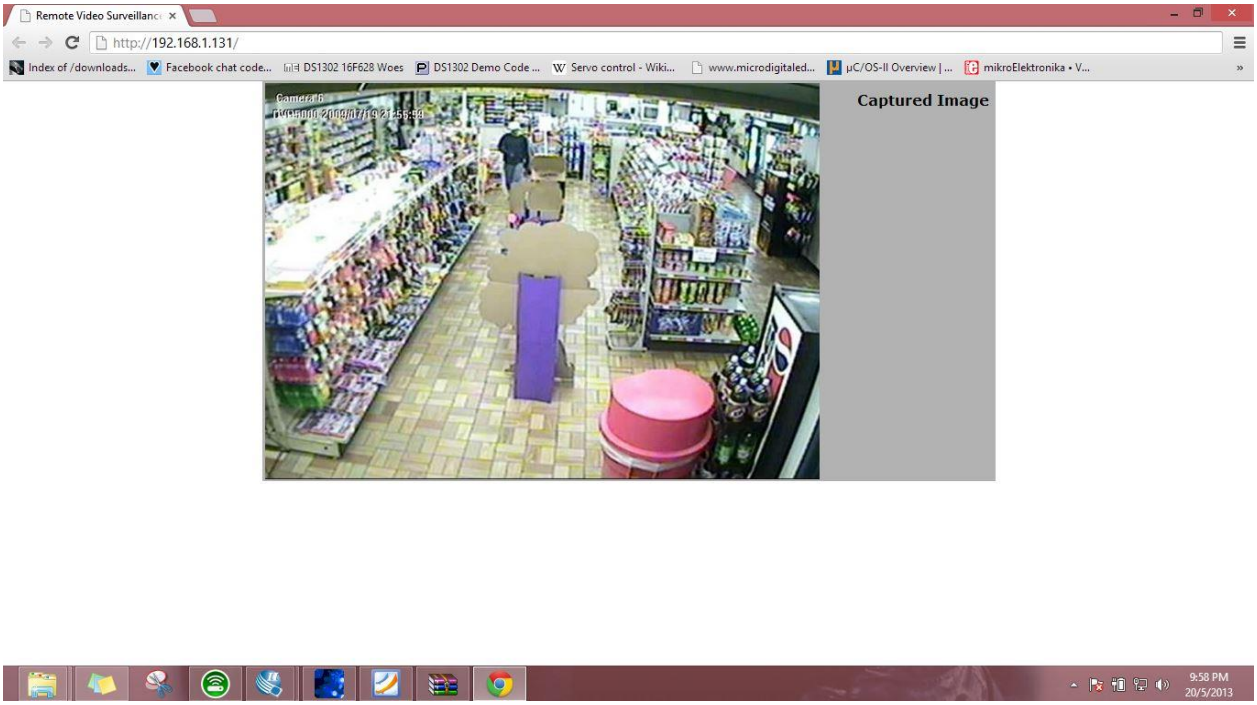


Figure 4.21: Image Display on Browser

This will be the result when the image access is a success

4.5 Problem and Troubleshooting

The result cannot be displayed as the project stumbled on an error during the building of project in Nios II IDE. The error is as follows:

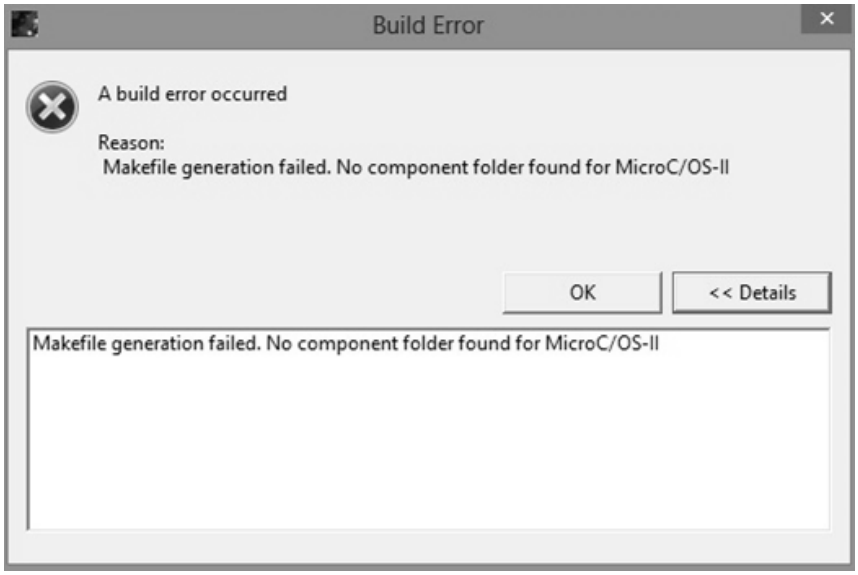

Build Error
A build error occurred
Reason:
Makefile generation failed. No component folder found for MicroC/OS-II
OK
<< Details
Makefile generation failed. No component folder found for MicroC/OS-II

Figure 4.22: MicroC/OS-II Missing Component

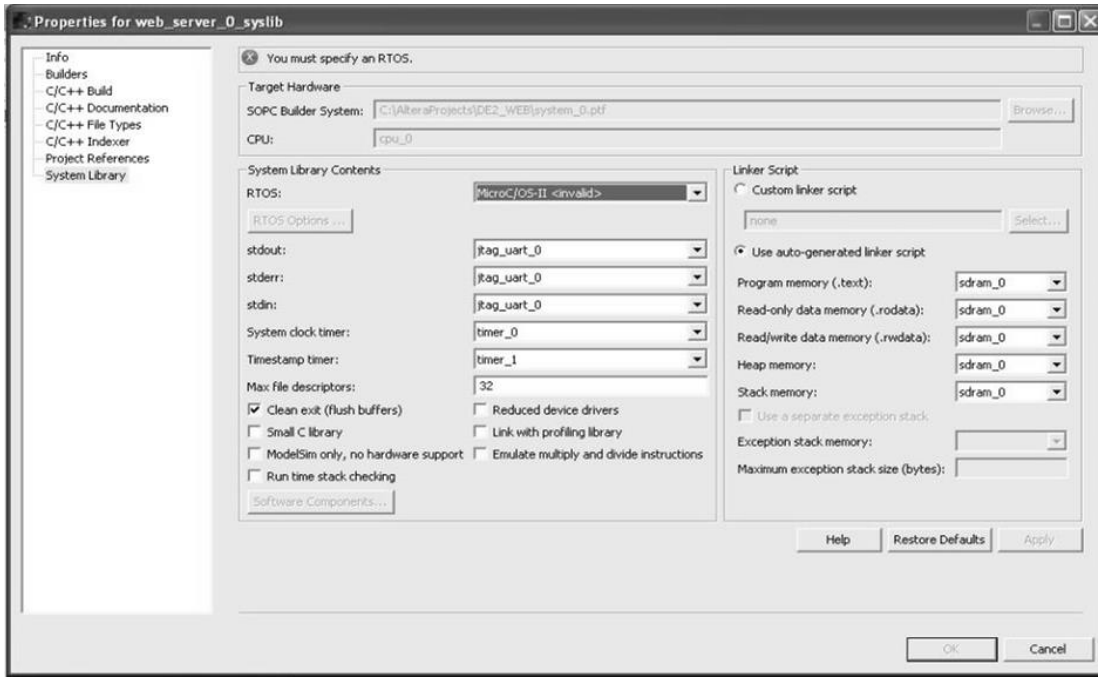


Figure 4.23: System Library Properties

When the project is built, the makefile generation failed as the component required to run the program is missing and unavailable. The system library properties also flags the same error, stating that the MicroC/OS-II component is missing and invalid. MicroC/OS-II is an abbreviation of Micro Controller Operating System

version 2. It is a small real-time kernel, mainly written in C language which is intended for use in embedded systems. In this project, this component is required to run the web server type system. MicroC/OS-II or uCOS-II is part of Real Time Operating System (RTOS). RTOS is an operating system intended to serve real-time application request.

This problem related to software problem, so Altera was contacted regarding this problem through e-mail. Altera engineers gave their insight regarding this problem, that uCOS-II component is missing due to Nios II software not installed properly, and the current software might be "broken". So the software was re-downloaded at Altera ftp site, and Nios II software was reinstalled. However, this did not solve the problem.

Altera was contacted again but this time the engineers suspects the IP core might have some problem and not the software. Since the IP core was provided by TERASIC, the problem was re-directed to TERASIC for troubleshooting. TERASIC engineer suggested a method to solve the problem. The project was recreated under the web server template, which is available during the project creation in Nios II. The coding in c language and the image file in zip format was copied into the new project. The project was built but the problem still persists. TERASIC was contacted again and the engineers suggested that Altera should be consulted regarding this problem as this problem was due to software error.

Altera was contacted several times regarding this issue, but no response from the engineers regarding this matter. This problem was then being discussed in the Altera Forum. There are some suggestions given and tried out, which are regenerate the SOPC system file, and recompile the project. Unfortunately, the method did not work out. The uCOS-II currently provided by Micrium. So the uCOS-II was searched at the website but there are no uCOS-II component or the current board used for the project, which are cyclone II type board. The available component is for cyclone III board. Nevertheless, the component was downloaded but the integration failed.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

This project purpose of creating the network module for remote video surveillance can be considered a success as all the necessary soft-hardware creations, image file format creations, and software (coding) creations was completed. This project has achieved the objective of creating an image file and embedded it in an html file where the html files and the images are stored in a zip format before successfully being stored in the flash memory. The protocol creation/ the IP core was created and integrated successfully in the code. However, the objective of viewing the image stored inside the flash memory was unsuccessful due to Nios II software problem of not having the required uCOS-II component to run the web server type project. Even though there are missing component, the build of the project does not show any error on the code. So the coding was a success, and the project aim of creating a web server type system is a success. In this project, the computer should be able to access the image stored inside the FPGA DE2 flash memory by requesting through a browser using a specific IP address. The server responds and process the request to complete the overall flow of the project.

5.2 Recommendations

This project is Remote Video Surveillance using FPGA. The scope of this project was limited to the creation of the protocol and method of communications between FPGA and computer, so that the computer can access the image. The scope also involves the storing of image file into the flash memory. This project can be further improvised by adding the camera module. Creating the camera module was not included in the scope of this project as the creation need to include the utilization of Phillips ISP1362 USB controller chip inside FPGA DE2. A protocol for USB communication using this chip must be created and it will take a long time to develop. Creating this protocol for USB communications, and storing the image taken into the flash memory will create a complete-standalone video surveillance system. However, in order to create a real and more powerful video surveillance system, live video feed can be used to replace the image. This kind of design/implementation should allow any user connected to the network to access and view the video streamed by the camera. To add more security to the system, a password can be created when user tries to access using the browser. Only if the password is correct, then the video stream can be accessed.

The support provided by Altera was not efficient and fast. The response from the Altera and their engineers can be improved, as the project can achieve success and further improvisations if the software problem has been solved.

REFERENCE

An Braeken , J. G., Serge Kubera, Nele Mentensyz, Abdellah Touhafi, Ingrid Verbauwhedez,Yannick Verbelen, Jo Vliegenyz, Karel Woutersz (2011). "Secure Remote Reconfiguration of an FPGA-based Embedded System." 1-6.

Brain, M. "How USB Ports Work." from http://computer.howstuffworks.com/usb1.htm.

Corporation, A. (2006). DE2 Development and Education Board,User Manual**:** 7.

Corporation, A. (2006). DE2 Development and Education Board,User Manual**:** 9.

Corporation, A. (2006). DE2 Development and Education Board,User Manual**:** 46.

Corporation, A. (2012). "Nios II Processor: The World's Most Versatile Embedded Processor." from http://www.altera.com.my/devices/processor/nios2/ni2-index.html.

Inc, D. S. (2006). DM9000A Ethernet Controller with General Processor Interface.

Ping Xue, J. H., Haichao Wang and Jianwei Ma (2011). "The Design of Remote Image Monitoring System Based on DM9000A." 885-889.

Rami Amiri, O. E. (2012). "A Reconfigurable Hardware Networking Platform for Smart Grid." 1-4.

Semiconductor, P. (2002). Philips ISP1362, Koninklijke Philips Electronics.

Staff, T. S. G. (2012). "EC unveils plan to allow wireless technologies, including broadband, to share radio spectrum." Business, Economy, EU, News. from http://sofiaglobe.com/2012/09/03/ec-unveils-plan-to-allow-wireless-technologies-including-broadband-share-radio-spectrum/.

Wikipedia. "Internet protocol suite." from http://en.wikipedia.org/wiki/Internet_protocol_suite.

Corporation, A. (2011). "Using the NicheStack TCP/IP Stack - Nios II Edition." 1-36.

Pidgeon, N. "How Ethernet Works." from http://computer.howstuffworks.com/ethernet4.htm.

Technologies, T. "NIOS II Web Server Demo." 1-6.

Microsoft. "TCP/IP Core Protocols." from http://technet.microsoft.com/en-us/library/cc958827.aspx.

"Nios II." from http://en.wikipedia.org/wiki/Nios_II.

APPENDIX A

FYP 1 Gantt Chart

	Weeks														
Activities	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
FYP Briefing															
Methodology Briefing															
FYP Title searching															
Searching materials related to FYP and literature review															
Write project proposal and submit															
Progress report FYP 1-2															
Progress report FYP 1-3															
Preparation for Seminar FYP 1															
Presentation of Seminar FYP 1															
Preparation for FYP 1 Report															
FYP 1 Report Submission															

APPENDIX B

FYP 2 Gantt Chart

	Weeks															
Activities	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Creating image file and store in flash memory																
Developing protocol for data/packets transfer																
Accessing image file stored in DE2 board flash from pc																
Testing and trouble shooting																
Meeting with supervisor																
Preparation for Seminar FYP 2																
Presentation of Seminar FYP 2																
Preparation for FYP 2 Report																
FYP 2 Report Submission																

APPENDIX C

Nios II Coding

web_server.c

```
#include <stdio.h>
#include <errno.h>
#include <ctype.h>
#include "includes.h"
#include "alt_lwip_dev.h"
#include "lwip/sys.h"
#include "user.h"

#include "dm9000.h"
#include "lcd.h"
ALTERA_AVALON_DM9K_INSTANCE(DM9000A, dm9k);

void user_task(void * pvoid)
{
  static u_long val=0;

  // simply doing sanity check
  for(;;)
  {
    val ^= (1<<17);
    IOWR(LED_RED_BASE, 0, val);
    usleep(500000);
  } // of forever loop
}

#ifndef LWIP
  #error This Web Server requires the Lightweight IP Software Component.
#endif

#ifndef __ucosii__
  #error This Web Server requires the UCOS II IP Software Component.
#endif

#ifndef  RO_ZIPFS
  #error This Web Server requires the Altera Read only Zip filing system.
#endif

/*
 * A MicroC/OS-II message box will be used to communicate between telnet
 * and board LED control tasks.
 */
OS_EVENT *attained_ip_address_sem;

/*
 * tcpip_init_done()
 *
 * This function is called once the LWIP stack is alive, as specified in our
 * lwip_stack_init() call in main(), below.
 *
 * We tell it about each of our network devices and start the application &
 * timer threads as appropriate.
 */
static void tcpip_init_done(void *arg)
{

  ALTERA_AVALON_DM9K_INIT(dm9k);

  /*
   * At this point LWIP has been initialized, but the Ethernet interface has
   * not; the initialise_lwip_devices() call does so, adding in MicroC-OS/II
   * threads for low-level Ethernet MAC interface and TCP protocol timer.
   */
  if (!lwip_devices_init(ETHER_PRIO))
    die_with_error("[tcpip_init_done] Fatal: Can't add ethernet interface!");

    attained_ip_address_sem = OSSemCreate(1);
```

```
#if LWIP_DHCP == 1

  if(!(IORD(SWITCH_PIO_BASE, 0) & (1<<17))) sys_thread_new(dhcp_timeout_task, NULL, DHCP_TMR_PRIO);


#endif /* LWIP_DHCP */

  /*
   * Add any application task(s) that rely on LWIP:
   *
   * http_task() -- Creates HTTP listening sockets, manages incoming
   *                connection requests, and handles all web server duties.
   */
  if(!sys_thread_new(http_task, NULL, HTTP_PRIO))
    die_with_error("[tcpip_init_done] Fatal: Can't add HTTP task!");

}

int main ()
{

  LCD_Init();

  //printf("\n\nWeb Server starting up\n\n");

  /*
   * Our very first call in an LWIP example design is to start up the LWIP
   * stack. We specify a MicroC/OS-II thread priority for the tcp_ip thread,
   * as well as a call-back routine, tcpip_init_done(), which is called once
   * the stack is alive and well.
   */
  lwip_stack_init(TCPIP_PRIO, tcpip_init_done, 0);
  /*
   * As with all MicroC/OS-II designs, once the initial thread(s) and
   * associated RTOS resources are declared, we start the RTOS. That's it!
   */


  sys_thread_new(user_task, NULL, SANITY_PRIO);

  OSStart();

  return 0;
}
```

network_utilities.c

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include "alt_lwip_dev.h"
#include "alt_types.h"
#include "sys/alt_flash.h"
#include "includes.h"
#include "io.h"
#include "user.h"

#include "lcd.h"
#ifdef LWIP_DEFAULT_IF
#undef LWIP_DEFAULT_IF
#define LWIP_DEFAULT_IF "DM9000A"
#endif

#ifdef ALT_DEBUG
#define VERB(msg) printf msg
#else
#define VERB(msg) do { } while (0)
#endif

/*
 * The "adapter" storage here is declared so that when LWIP calls the
 * get_ip_addr() routine, the adapter struct will be populated with the
 * network device (Ethernet MAC) that LWIP is currently trying to setup an IP
 * address for. This is later used in the dhcp_timeout_task() routine to
 * determine if DHCP was successful for the particular adapter.
 */
struct netif* adapter;
```

```
extern OS_EVENT *attained_ip_address_sem;

void die_with_error(char err_msg[DIE_WITH_ERROR_BUFFER])
{
  printf("\n%s\n", err_msg);
  OSTaskDel(OS_PRIO_SELF);

  while(1);
}

int get_ip_addr(alt_lwip_dev* lwip_dev, struct ip_addr* ipaddr,
                struct ip_addr* netmask, struct ip_addr* gw,
                int* use_dhcp)
{
  int ret_code = 0;

  int sw;
  struct netif* netif = lwip_dev->netif;

  sw = IORD(SWITCH_PIO_BASE, 0);

  if(sw & (1<<17))
  {
    char buf[32];
    char buf1[32];
    u_char ip3, ip4;
    u_char * ptr;

    *use_dhcp = 0;

    ip3 = (sw >> 12) & 0x0F;
    ip4 = ((sw >> 8) & 0x0F) + 128;
    sprintf(buf, "#192.168.%d.%d", ip3, ip4);
    LCD_Show_Text(buf);
    LCD_Line2();
    sprintf(buf1, "MAC:009000AE00%2X",netif->hwaddr[5]);
    LCD_Show_Text(buf1);

    VERB(("Use static IP configuration, IP = %s\n", buf));

    ptr = (u_char*)ipaddr;
    *ptr++ = 192;
    *ptr++ = 168;
    *ptr++ = ip3;
    *ptr++ = ip4;

    ptr = (u_char*)netmask;
    *ptr++ = 0xff;
    *ptr++ = 0xff;
    *ptr++ = 0xff;
    *ptr++ = 0;

    ptr = (u_char*)gw;
    *ptr++ = 192;
    *ptr++ = 168;
    *ptr++ = ip3;
    *ptr++ = 1;

    return 1;
  } // of use static

  if (!strcmp(lwip_dev->name, "/dev/" LWIP_DEFAULT_IF))
  {
#if LWIP_DHCP == 1
    *use_dhcp = 1;
    adapter = lwip_dev->netif;

#else
    IP4_ADDR(ipaddr, IPADDR0,IPADDR1,IPADDR2,IPADDR3);
    IP4_ADDR(gw, GWADDR0,GWADDR1,GWADDR2,GWADDR3);
    IP4_ADDR(netmask, MSKADDR0,MSKADDR1,MSKADDR2,MSKADDR3);
    *use_dhcp = 0;

#endif /* LWIP_DHCP */


    ret_code = 1;
  }

  return ret_code;
}
```

```
#if LWIP_DHCP == 1
void dhcp_timeout_task()
{
  int have_address = 0, timeout_dhcp = 0;
  struct ip_addr ipaddr, netmask, gw;
  INT8U return_code = OS_NO_ERR;

  OSSemPend(attained_ip_address_sem, 0, &return_code);

  while(1)
  {
    OSTimeDlyHMSM(0,0,0,100);

    if((timeout_dhcp < DHCP_TIMEOUT) && (have_address == 0))
    {
      timeout_dhcp++;
      if (!ip_addr_isany(&adapter->ip_addr))
      {
        have_address = 1;

        VERB(("Assigned IP Address is %d.%d.%d.%d\n\n",ip4_addr1(&adapter->ip_addr),
        ip4_addr2(&adapter->ip_addr),ip4_addr3(&adapter->ip_addr),ip4_addr4(&adapter->ip_addr)));
        VERB(("Assigned GW Address is %d.%d.%d.%d\n\n",ip4_addr1(&adapter->gw),
        ip4_addr2(&adapter->gw),ip4_addr3(&adapter->gw),ip4_addr4(&adapter->gw)));
        {
      char buf[32];
      sprintf(buf, "#%d.%d.%d.%d",
        ip4_addr1(&adapter->ip_addr),
        ip4_addr2(&adapter->ip_addr),
        ip4_addr3(&adapter->ip_addr),
        ip4_addr4(&adapter->ip_addr)
        );
        LCD_Show_Text(buf);
        LCD_Line2();

      sprintf(buf, "MAC:009000AE00%2X",
        adapter->hwaddr[5]
        );
        LCD_Show_Text(buf);
        }
      }
    }

    if (timeout_dhcp ==  DHCP_TIMEOUT)
    {
      dhcp_stop(adapter);
      IP4_ADDR(&ipaddr, IPADDR0,IPADDR1,IPADDR2,IPADDR3);
      IP4_ADDR(&gw, GWADDR0,GWADDR1,GWADDR2,GWADDR3);
      IP4_ADDR(&netmask, MSKADDR0,MSKADDR1,MSKADDR2,MSKADDR3);

      netif_set_addr( adapter,&ipaddr, &netmask, &gw);
      netif_set_up(adapter);

      have_address = 1;
    }

    if (have_address == 1)
    {

       OSSemPost(attained_ip_address_sem);
       OSTaskDel(OS_PRIO_SELF);
    }
  }
}
#endif /* LWIP_DHCP */
```

http.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <sys/param.h>
#include <sys/fcntl.h>
#include "lwip/netif.h"
#include "lwip/sockets.h"
#include "arch/sys_arch.h"
#include "sys/alt_alarm.h"
#include "alt_types.h"
#include "http.h"
#include "user.h"


#if 0
struct __sFILE {
  unsigned char *_p;  /* current position in (some) buffer */
  int _r;   /* read space left for getc() */
  int _w;   /* write space left for putc() */
  short _flags;   /* flags, below; this FILE is free if 0 */
  short _file;    /* fileno, if Unix descriptor, else -1 */
  struct __sbuf _bf;  /* the buffer (at least 1 byte, if !NULL) */
  int _lbfsize; /* 0 or -_bf._size, for inline putc */

  /* operations */
  _PTR  _cookie;  /* cookie passed to io functions */

  _READ_WRITE_RETURN_TYPE _EXFUN((*_read),(_PTR _cookie, char *_buf, int _n));
  _READ_WRITE_RETURN_TYPE _EXFUN((*_write),(_PTR _cookie, const char *_buf,int _n));
  _fpos_t _EXFUN((*_seek),(_PTR _cookie, _fpos_t _offset, int _whence));
  int _EXFUN((*_close),(_PTR _cookie));
  /* separate buffer for long sequences of ungetc() */
  struct __sbuf _ub;  /* ungetc buffer */
  unsigned char *_up; /* saved _p when _p is doing ungetc data */
  int _ur;    /* saved _r when _r is counting ungetc data */

  /* tricks to meet minimum requirements even when malloc() fails */
  unsigned char _ubuf[3]; /* guarantee an ungetc() buffer */
  unsigned char _nbuf[1]; /* guarantee a getc() buffer */

  /* separate buffer for fgetline() when line crosses buffer boundary */
  struct __sbuf _lb;  /* buffer for fgetline() */

  /* Unix stdio files get aligned to block boundaries on hyseek() */
  int _blksize; /* stat.st_blksize (may be != _bf._size) */
  int _offset;  /* current lseek offset */

#ifndef __SINGLE_THREAD__
  __flock_t _lock; /* for thread-safety locking */
#endif
};
#endif

#define MAX_CONCURRENT_HYPER  64
#define HYPER_FID_START     0x7000
FILE  _gFiles[MAX_CONCURRENT_HYPER];
int _ghy_initd = 0;

#define HY_CHK_FP(fp) ((fp) && ((fp)->_file >= HYPER_FID_START))

FILE * hy_allocate_fp(void)
{
  int i;
  for(i=0;i<MAX_CONCURRENT_HYPER;i++)
  {
    if(_gFiles[i]._flags != 0) continue;
    _gFiles[i]._flags = 0xbeef;
    return &_gFiles[i];
  } // of for i

  return NULL;
}

void hy_release_fp(FILE *fp)
{
  fp->_flags = 0;
}
```

```
FILE * hyopen(const char * pfname, const char * pmode)
{
  // bypass if ordinary file
  if(strncmp(pfname+strlen(ALTERA_RO_ZIPFS_NAME), "/file?", 5)) return fopen(pfname, pmode);

  // initialization
  if(!_ghy_initd)
  {
    int i;
    _ghy_initd = 1;
    for(i=0;i<MAX_CONCURRENT_HYPER;i++)
    {
      _gFiles[i]._file = HYPER_FID_START + i;
      _gFiles[i]._flags = 0;
    } // of for i
  } // of not init'd

  FILE * fp;

  if(!(fp = hy_allocate_fp())) return NULL;

  int flen=0;
  char * ptr;

  // seek for size
  if((ptr=strstr(pfname, "size="))) flen = atoi(ptr+5);

  // seek for unit
  for(ptr+=5; *ptr ; ptr++)
    if(*ptr < '0' || *ptr > '9') break;

  switch (*ptr)
  {
    case 'k': // 1000
      flen *= 1000;
      break;
    case 'K': // 1024
      flen *= 1024;
      break;
    case 'm': // 1,000,000
      flen *= 1000000;
      break;
    case 'M': // 1024*1024
      flen *= 1024*1024;
      break;
    default:
      break;
  } // of switch

  fp->_cookie = flen ? (void*)flen:(void*)(1024*1024);

  return fp;
}

int hygetpos(FILE * fp, fpos_t * offset)
{
  if(!HY_CHK_FP(fp)) return fgetpos(fp, offset);

  *offset = (int)fp->_offset;

  return 0;
}
int hyseek(FILE * fp, long offset, int origin)
{
  if(!HY_CHK_FP(fp)) return fseek(fp, offset, origin);

  switch(origin)
  {
    case SEEK_SET:
      fp->_offset = offset;
      break;

    case SEEK_CUR: // is it right ?
      fp->_offset += offset;
      break;

    case SEEK_END:
      fp->_offset = (int)(fp->_cookie) + offset;
      break;

    default:
      return -1;
  } // of switch(origin)

  return 0;
}
```

```
int hyclose(FILE * fp)
{
  if(!HY_CHK_FP(fp)) return fclose(fp);

  hy_release_fp(fp);

  return 0;
}

int hyread(void * pbuffer, size_t size, size_t count, FILE * fp)
{
  if(!HY_CHK_FP(fp)) return fread(pbuffer, size, count, fp);

  int total, remains;

  remains = (int)(fp->_cookie) - fp->_offset;
  total = (remains >= size*count)? size*count:remains;
  fp->_offset += total;
  return total;
}


#ifdef DEBUG1
  #include alt_debug.h
#else
  #define ALT_DEBUG_ASSERT(a)
#endif /* DEBUG */

alt_u8 http_rx_buffer[HTTP_NUM_CONNECTIONS][HTTP_RX_BUF_SIZE];
alt_u8 http_tx_buffer[HTTP_NUM_CONNECTIONS][HTTP_TX_BUF_SIZE];
static const alt_8 canned_http_response[] = {"\
HTTP/1.0 404 Not Found\r\n\
Content-Type: text/html\r\n\
Content-Length: 272\r\n\r\n\
<HTML><HEAD><TITLE>Nios II Web Server Demonstration</TITLE></HEAD>\
<title>LWIP on Nios II</title><BODY><h1>HTTP Error 404</h1>\
<center><h2>Nios II Web Server Demonstration</h2>\
Can't find the requested file file. \
Have you programmed the flash filing system into flash?</html>\
"};

static const alt_8 hello_http_response[] = {"\
HTTP/1.0 200\r\n\
Content-Type: text/html\r\n\
Content-Length: 203\r\n\r\n\
<HTML><HEAD><TITLE>Nios II Web Server Demonstration</TITLE></HEAD>\
<title>LWIP on Nios II</title>\
<BODY>\
<br><br>\
<center>\
<h1>Nios II Web Server Demonstration</h1><br>\
<h2>Hello World</h2><br>\
</center>\
</html>\
"};

typedef struct funcs
{
  alt_u8*  name;
  void (*func)();
}post_funcs;
void print()
{
  printf("HTTP POST received.\n");
}

post_funcs mapping =
{
  "/PRINT",
  print
};

void http_reset_connection(http_conn* conn, int http_instance)
{
  memset(conn, 0, sizeof(http_conn));

  conn->fd = -1;
  conn->state = READY;
  conn->keep_alive_count = HTTP_KEEP_ALIVE_COUNT;

  conn->rx_buffer = (alt_u8 *) &http_rx_buffer[http_instance][0];
  conn->tx_buffer = (alt_u8 *) &http_tx_buffer[http_instance][0];
  conn->rx_wr_pos = (alt_u8 *) &http_rx_buffer[http_instance][0];
  conn->rx_rd_pos = (alt_u8 *) &http_rx_buffer[http_instance][0];
}
```

```
void http_manage_connection(http_conn* conn, int http_instance)
{
  alt_u32 current_time = 0;

  if(conn->state == READY || conn->state == PROCESS || conn->state == DATA)
  {
    current_time = alt_nticks();

    if( (current_time - conn->activity_time) >= HTTP_KEEP_ALIVE_TIME )
    {
      conn->state = RESET;
    }
  }

  if(conn->state == COMPLETE)
  {
    if(conn->file_handle != NULL)
    {
      hyclose(conn->file_handle);
    }

    conn->keep_alive_count--;
    conn->data_sent = 0;

    if(conn->keep_alive_count == 0)
    {
      conn->close = 1;
    }

    conn->state = conn->close ? CLOSE : READY;
  }
  if(conn->state == RESET)
  {
    if(conn->file_handle != NULL)
    {
      hyclose(conn->file_handle);
    }

    memset(conn->rx_buffer, 0, HTTP_RX_BUF_SIZE);
    conn->state = CLOSE;
  }

  /* Close the TCP connection */
  if(conn->state == CLOSE)
  {
    close(conn->fd);
    http_reset_connection(conn, http_instance);
  }
}

int http_handle_accept(int listen_socket, http_conn* conn)
{
  int ret_code = 0, i, socket, len;
  struct sockaddr_in  rem;

  len = sizeof(rem);

  for(i=0; i<HTTP_NUM_CONNECTIONS; i++)
  {
    if((conn+i)->fd == -1)
    {
      break;
    }
  }

  if(i == HTTP_NUM_CONNECTIONS)
    return -1;

  if((socket = accept(listen_socket,(struct sockaddr*)&rem,&len)) < 0)
  {
    //fprintf(stderr, "[http_handle_accept] accept failed (%d)\n", socket);
    return socket;
  }

  (conn+i)->fd = socket;
  (conn+i)->activity_time = alt_nticks();

  return ret_code;
}
```

```
int http_read_line(http_conn* conn)
{
  alt_u8* lf_addr;
  int ret_code = 0;

  /* Find the Carriage return which marks the end of the header */
  lf_addr = strchr(conn->rx_rd_pos, '\n');

  if (lf_addr == NULL)
  {
    ret_code = -1;
  }
  else
  {

    if ((lf_addr > conn->rx_buffer) && (*(lf_addr-1) == '\r'))
    {
      *(lf_addr-1) = 0;
    }

    *lf_addr = 0;
    conn->rx_rd_pos = lf_addr+1;
  }

  return ret_code;
}

int http_process_headers(http_conn* conn)
{
  alt_u8* option;
  alt_u8* orig_read_pos = conn->rx_rd_pos;
  alt_u8* temp;
  if( (temp = strstr(conn->rx_rd_pos, ":")) )
  {
    conn->rx_rd_pos = temp + 1;
  }
  else
  {
    return -1;
  }

  option = strtok(orig_read_pos, ":");

  if(stricmp(option,"Connection") == 0)
  {
    alt_u8 temp = *(option + 17);
    *(option + 17) = 0;

    if(stricmp((option+12), "close") == 0)
    {
      conn->close = 1;
    }
    *(option + 17) = temp;
  }
  else if (stricmp(option, "Content-Length") == 0)
  {
   conn->content_length = atoi(option+16);
  }

  return 0;
}
int http_process_request(http_conn* conn)
{
  alt_u8* uri = 0;
  alt_u8* version = 0;
  alt_u8* temp = 0;
  if( (temp = strstr(conn->rx_rd_pos, "GET")) )
  {
    conn->action = GET;
    conn->rx_rd_pos = temp;
  }
  else if( (temp = strstr(conn->rx_rd_pos, "POST")) )
  {
    conn->action = POST;
    conn->rx_rd_pos = temp;
  }
  else
  {
    //fprintf(stderr, "Unsupported (for now) request\n");
    conn->action = UNKNOWN;
    return -1;
  }
```

```
  while(1)
  {
    FD_ZERO(&readfds);
    FD_ZERO(&writefds);
    FD_SET(fd_listen, &readfds);

    max_socket = fd_listen+1;

    for(i=0; i<HTTP_NUM_CONNECTIONS; i++)
    {
      if (conn[i].fd != -1)
      {
        /* We're interested in reading any of our active sockets */
        FD_SET(conn[i].fd, &readfds);

        /*
         * We're interested in writing to any of our active sockets in the DATA
         * state
         */
        if(conn[i].state == DATA)
        {
          FD_SET(conn[i].fd, &writefds);
        }

        if (max_socket <= conn[i].fd)
        {
          max_socket = conn[i].fd+1;
        }
      }
    }

    select_timeout.tv_sec = 0;
    select_timeout.tv_usec = 500000;

    select(max_socket, &readfds, &writefds, NULL, &select_timeout);

    if (FD_ISSET(fd_listen, &readfds))
    {
      http_handle_accept(fd_listen, conn);
    }

    for(i=0; i<HTTP_NUM_CONNECTIONS; i++)
    {
      if (conn[i].fd != -1)
      {
        if(FD_ISSET(conn[i].fd,&readfds))
        {
          http_handle_receive(&conn[i], i);
        }

        if(FD_ISSET(conn[i].fd,&writefds))
        {
          http_handle_transmit(&conn[i], i);
        }

        http_manage_connection(&conn[i], i);
      }
    }
  } /* while(1) */
}
```

lcd.c

```
#include <unistd.h>
#include <string.h>
#include <io.h>
#include "system.h"
#include "LCD.h"
//-----------------------------------------------------------------------
void LCD_Init()
{
  lcd_write_cmd(LCD_16207_0_BASE,0x38);
  usleep(2000);
  lcd_write_cmd(LCD_16207_0_BASE,0x0C);
  usleep(2000);
  lcd_write_cmd(LCD_16207_0_BASE,0x01);
  usleep(2000);
  lcd_write_cmd(LCD_16207_0_BASE,0x06);
  usleep(2000);
  lcd_write_cmd(LCD_16207_0_BASE,0x80);
  usleep(2000);
}
//-----------------------------------------------------------------------
void LCD_Show_Text(char* Text)
{
  int i;
  for(i=0;i<strlen(Text);i++)
  {
    lcd_write_data(LCD_16207_0_BASE,Text[i]);
    usleep(2000);
  }
}
//-----------------------------------------------------------------------
void LCD_Line2()
{
  lcd_write_cmd(LCD_16207_0_BASE,0xC0);
  usleep(2000);
}
//-----------------------------------------------------------------------
void LCD_Test()
{
  char Text1[16] = "<NIOS II on DE2>";
  char Text2[16] = "Nice to See You!";
  //  Initial LCD
  LCD_Init();
  //  Show Text to LCD
  LCD_Show_Text(Text1);
  //  Change Line2
  LCD_Line2();
  //  Show Text to LCD
  LCD_Show_Text(Text2);
}
//-----------------------------------------------------------------------
```

dm9000.c

```
#define USE_ARP_SEMAPHORE

#include "dm9000.h"

#ifdef ALT_DEBUG
#define VERB(msg) printf msg
#else
#define VERB(msg) do { } while (0)
#endif

inline void dm9k_idxout(alt_avalon_dm9k_if * pdev, u_char val)
{
  IOWR(pdev->base_addr, pdev->index_offset, val);
}

inline u_char dm9k_idxin(alt_avalon_dm9k_if * pdev)
{
  return (u_char)(IORD(pdev->base_addr, pdev->index_offset) & 0xff);
}

inline void dm9k_datout(alt_avalon_dm9k_if * pdev, u_char val)
{
  IOWR(pdev->base_addr, pdev->data_offset, val);
}

inline u_char dm9k_datin(alt_avalon_dm9k_if * pdev)
{
  return (u_char)(IORD(pdev->base_addr, pdev->data_offset) & 0xff );
}

inline void dm9k_datoutw(alt_avalon_dm9k_if * pdev, u_short val)
{
  IOWR(pdev->base_addr, pdev->data_offset, val);
}
inline u_short dm9k_datinw(alt_avalon_dm9k_if * pdev)
{
  return (u_short)(IORD(pdev->base_addr, pdev->data_offset) & 0xffff );
}

inline void dm9k_datoutl(alt_avalon_dm9k_if * pdev, u_long val)
{
  IOWR(pdev->base_addr, pdev->data_offset, val);
}

inline u_long dm9k_datinl(alt_avalon_dm9k_if * pdev)
{
  return (u_long)(IORD(pdev->base_addr, pdev->data_offset));
}

inline u_char dm9k_regin(alt_avalon_dm9k_if * pdev, u_char regidx)
{
  dm9k_idxout(pdev, regidx);
  return dm9k_datin(pdev);
}

inline void dm9k_regout(alt_avalon_dm9k_if * pdev, u_char regidx, u_char val)
{
  dm9k_idxout(pdev, regidx);
  dm9k_datout(pdev, val);
}

inline void dm9k_meminw(alt_avalon_dm9k_if * pdev, u_short * pbuffer, int len)
{
  dm9k_idxout(pdev, 0xf2);
  for(len=(len+1)/2;len--;) *pbuffer++ = dm9k_datinw(pdev);
}
```

```
int dm9k_identify(alt_avalon_dm9k_if * pdev)
{
  // vid(29:28) = 0A:46
  if((dm9k_regin(pdev, 0x28)!= 0x46)||(dm9k_regin(pdev,0x29)!=0x0a))return -1;
  // pid(2B:2A) = 90:00
  if((dm9k_regin(pdev, 0x2A)!= 0x00)||(dm9k_regin(pdev,0x2B)!=0x90))return -1;

  return 0;
}


err_t alt_avalon_dm9k_init(struct netif *netif)
{
  int i;
  alt_avalon_dm9k_if * pdev = (alt_avalon_dm9k_if*)netif->state;
  pdev->lwip_dev_list.dev.netif = netif;

  // check private data's availability
  if (!pdev) return ERR_IF;

  // identify our device
  if(dm9k_identify(pdev) == -1) return ERR_IF;

  // specify callback routines
  netif->name[0] = 'd';
  netif->name[1] = 'm';
  netif->output = dm9k_output;
  netif->linkoutput = dm9k_link_output;

  // set physical address
  //
  pdev->hwaddr[5] = IORD(SWITCH_PIO_BASE, 0) & 0xFF;
  for(i=0;i<(netif->hwaddr_len = 6);i++) netif->hwaddr[i] = pdev->hwaddr[i];
  // dm9k initialization
  if(dm9k_init(pdev) != ERR_OK ||
    !(pdev->arp_semaphore = sys_sem_new(1)) ||
    !(pdev->tx_semaphore = sys_sem_new(pdev->dm9k_tx_space))
    )
  {
    free(netif->state);
    return ERR_IF;
  } // of dm9k initialization

  etharp_init();

  return ERR_OK;
}

void alt_avalon_dm9k_rx(alt_lwip_dev * paltdev)
{
  alt_avalon_dm9k_if * pdev = (alt_avalon_dm9k_if*)(paltdev->netif->state);

  // save current index register
  volatile u_char idxold = dm9k_idxin(pdev);

  for(;dm9k_input(paltdev->netif););

  // resotre old idx register
  dm9k_idxout(pdev, idxold);
}
  // 1. power on internal PHY
  dm9k_regout(pdev, 0x1f, 0x00);
  usleep(5000);

  // 2. software reset
  dm9k_regout(pdev, 0x00, 0x03);
  usleep(5000);

  // 3. clear Network Stautus Register(NSR) by reading out
  dm9k_regin(pdev, 0x01);

  // 4. set MAC address
  for(i=0;i<6;i++) dm9k_regout(pdev, 0x10+i, pdev->hwaddr[i]);

  // 4.1 set hash table
  // for(i=0;i<6;i++) dm9k_regout(pdev, 0x16+i, 0xff);

  // 5. clear Interrupt Status Register (NSR)
  dm9k_regout(pdev, 0xfe, 0x00);
```

```
err_t dm9k_link_output(struct netif *netif, struct pbuf *p)
{
  alt_avalon_dm9k_if * pdev = (alt_avalon_dm9k_if*)netif->state;

  // wait tx resource availability
  sys_sem_wait(pdev->tx_semaphore);

  // fill packet to MAC
  dm9k_memoutw(pdev, (u_short*)(p->payload), p->tot_len);

  // specify the length
  dm9k_regout(pdev, 0xfc, (u_char)(p->tot_len & 0xff));
  dm9k_regout(pdev, 0xfd, (u_char)((p->tot_len & 0xff00) >> 8));

  // kick start
  dm9k_regout(pdev, 0x02, 0x01);

  return ERR_OK;
}

int dm9k_input(struct netif *netif)
{
  struct eth_hdr  *ethhdr;
  struct pbuf     *p;
  alt_avalon_dm9k_if * pdev = (alt_avalon_dm9k_if*)netif->state;

  /* move received packet into a new pbuf */
  if(!(p = dm9k_link_input(pdev))) return 0;

  ethhdr = (struct eth_hdr*)(p->payload);

  switch (htons(ethhdr->type)) {
  /* IP packet? */
  case ETHTYPE_IP:
    /*
    * update ARP table
    */

    WAIT_ARP_SEMAPHORE(pdev);
    etharp_ip_input(netif, p);
    SIGNAL_ARP_SEMAPHORE(pdev);

    /* skip Ethernet header */
    pbuf_header(p, -(s16_t)sizeof(struct eth_hdr));

    /* pass to network layer */
    netif->input(p, netif);

    break;

  case ETHTYPE_ARP:
    /*
    * pass p to ARP module
    *
    */
    WAIT_ARP_SEMAPHORE(pdev);
    etharp_arp_input(netif, (struct eth_addr *)&netif->hwaddr, p);
    SIGNAL_ARP_SEMAPHORE(pdev);
    break;

  default:
    pbuf_free(p);
    p = NULL;
    break;
  } // of packet type switch

  return 1;
} // of alt_avalon_dm9k_rx

struct pbuf * dm9k_link_input(alt_avalon_dm9k_if * pdev)
{
  u_char saveidx;

  struct pbuf *p, *q;
  volatile u_short pklen, status;
  u_short remains;

  saveidx = dm9k_idxin(pdev);

  // check RX availability
  // first, dummy read
  dm9k_idxout(pdev, 0xf0);
  status = dm9k_datinw(pdev);
  // re-read again
  status = dm9k_datinw(pdev);
  if((status & 0xff) != 0x01)
  {
    dm9k_idxout(pdev, saveidx);
    return NULL;
  }
```

```
  // fill packet per chunk
  for(remains=pklen, q = p; remains && q; q = q->next)
  {
    dm9k_meminw(pdev, q->payload, q->len);
    remains -= q->len;
  } // of fill loop

  // finally, read out the 4-byte CRC
  dm9k_datinw(pdev);
  dm9k_datinw(pdev);

#if LINK_STATS
  lwip_stats.link.recv++;
#endif /* LINK_STATS */

  dm9k_idxout(pdev, saveidx);
  return p;

}
```